Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

NetStage: Web application for music event comparison and management

Estudante: Susana Rodríguez Lado
Dirección: Laura Milagros Castro Souto

A Coruña, febreiro de 2020.

For every musician

Acknowledgements

This would not have been possible without the help and patience of those teachers who were alongside me. I would like to particularly thank Laura for the guidance and support during this project, besides in her class. Of course, I'll thank my parents, who have always supported me. I'd finally like to thank every coffee break taken with Yolanda, and that particular day when I sat next to Saúl in class.

Abstract

This project developed an application that allows music event search and comparison, so the user can find events based on advanced criteria and follow events and artists of their interest to keep abreast of their published information. Administration and maintenance of this information will also be covered, allowing the event administrators and artists to update their data and event participation.

Resumo

Neste proxecto desenvolveuse unha aplicación que permite a búsqueda e comparación de eventos musicais, permitindo ao usuario comparacións en función de criterios avanzados, así como o seguimento de eventos e artistas do seu interese para manterse ao tanto da información que se publique e se actualice. Abárcase tamén a administración e mantemento de dita información por parte dos administradores de eventos e artistas, que poderán actualizar os seus datos e participación en eventos.

Keywords:

- Angular
- .Net
- Event comparison
- Web application
- Relational DB

Palabras chave:

- Angular
- .Net
- Comparación de eventos
- Aplicación web
- BD relacional

Contents

List of Figures

List of Tables

Chapter 1
Introduction

In this chapter the master lines of the project, motivation and objectives will be covered, along with an explanation of the current background in the music event industry to understand the motivations, which are tightly related.

1.1 Motivation

Nowadays, technology has made easy searching for a hotel or a flight, by using web comparators, but this does not apply to music events. There are many reasons for this; the different event organizers usually post their updates only through social media, making it impossible to find all the information at a centralized platform, and there are many more criteria to compare. Unlike finding a hotel or a flight, for choosing between two different festivals, you need to know, besides the location, date and price of these events, the artists involved in them, and the location is not as relevant as for hotels, where you have previously chosen the place you want to visit. In this case, the bands performing have much more relevance; the day tickets might not have all the bands you are trying to see, you might prefer to watch many artists of the genre you like rather than one big band, or you might only want to find a concert of your favourite band with the lowest price, as close to you as possible.

These kind of events have been growing since the beginning of the decade. In Spain, since 2005, we've experienced a 40% growth, with over a thousand festivals in total. In 2019 we hosted over 2.365.000 attendees only taking into account 23 of some of the biggest events celebrated [1]. Besides this kind of events are a common incentive for tourism, encouraging attendees not only to visit other towns, but also other countries.

For this amount of information, and given the complexity of the comparison criteria, a platform unifying this data and providing advanced algorithms would allow to find this kind of events in an easy, transparent way. With a platform like this, it would also be easy to provide followup of the events of interest for the users, allowing them to keep updated on the

Figure 1.1: Festival distribution in Spain. *Source: Ticketea, INE*

information published by the administration of the events, or the artists they're following. Unlike with social media, where the news of your favourite artists might get overwhelmed by other personal accounts posts forcing you to check up on every band and festival you might like, and not promote other festivals, this tool would provide you specific data on these events and artists avoiding the need to checkup on every one of them specifically.

1.2 Objectives

The main feature this project seeks to offer is a platform to unify the music event industry data, allowing external users to find and compare events, artists and news with the criteria mentioned before, avoiding the need of specific search through every event or artist page, and providing events of interest for the users that they might not find searching individually. For this purpose, the tool will offer the user two main kinds of search: Music event search and Artist search.

The **music event search** will allow filtering by artists participating in this events, location (by country and city), price (in case there are tickets available), date and genre (taking into account the main genres of the artists involved in the event).

The **artist search** will allow filtering by artist location (by country and city), and the genre of the artist or band.

As additional features, the user will be able to follow events and artists, so if new updates are provided by the administration, the users will have them listed in their news feed.

This application is mainly oriented to event attendees, but for it to work it will also be necessary to cover other user roles such as event administration and artist profile, so that the information gathered in the application can be managed by authorized personal. For this roles, the features offered will be the following:

The **Event administrator** will be able to manage several events, and for each one of them, their start and end date, location, and artist lineup, as well as ticket offers, official ticket sell point, social media and web-page, to ease access to those official sites. Attending artists and available tickets will also be managed, and custom notifications may be published in order to notify the ones selected by the administrator (relevant new headliners confirmed, or new ticket offers).

The **Artist profile** will allow artists to customize their profile page, setting their description, location, music genres and profile picture, among other data, and also manage and notify their attendance to events and concerts.

The sum of all this data will provide, not only an easy and unified point for comparison criteria for events and artists, but also persistent data in this matter, for future statistics and personalized suggestions for the user.

Chapter 2

Technological foundations

For the development of this application, many technologies were used in order to allow the planning, agile development, version control, etc. This chapter will list the main ones, grouping them in technologies used for back-end development and front-end development.

2.1 Back-end

The server side has been developed mainly through C#, .NET and SQL.

.NET

.NET[2][3] is an open source, cross-platform Framework developed by Microsoft which includes a large class library named as **Framework Class Library** (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (in contrast to a hardware environment) named the **Common Language Runtime** (CLR). FCL and CLR together constitute the .NET Framework.

CLR is an application virtual machine that provides services such as security, memory management, and exception handling. As such, computer code written using .NET Framework is called ”managed code”.

FCL provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their source code with .NET Framework and other libraries. The framework is intended to be used by most new applications created for the Windows platform. Microsoft also produces an integrated development environment largely for .NET software called Visual Studio.

.NET applications may be written in C#, F# or Visual Basic. For this application, C# has been chosen due to previous knowledge and experience on this language.

ASP.NET[4] extends the .NET platform with tools and libraries specifically for building modern web apps, so it was also used for this project. Some things that ASP.NET adds to the .NET platform are:

- Base framework for processing web requests.
- Web-page templating syntax, known as Razor, for building dynamic web pages using C#.
- Libraries for common web patterns, such as Model View Controller (MVC).
- Authentication system that includes libraries, a database, and template pages for handling logins, including multi-factor authentication and external authentication with Google, Twitter, and more.
- Editor extensions to provide syntax highlighting, code completion, and other functionality specifically for developing web pages.

SQL

SQL[5] is a domain-specific language used in programming and designed for managing data held in a Relational Database Management System (RDBMS), or for stream processing in a Relational Data Stream Management System (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

In this application Microsoft SQL Server[6] was used. DAOs access the database through SQL queries to retrieve data in a specific manner, allowing complex filtering criteria and formatted extraction of the information. An example of a SQL query can be found at Implementation chapter section 7.1

The Database creation script also included indexes[7], which are SQL rules which allow creating metadata for a faster data obtainment depending on a given attribute, among other features. For example, an index on a name attribute of a given entity provides metadata of names of the instances of the said entity ordered by name, so when an SQL query looks for

a specific one, they can be found easily. These indexes might be used to sort tables from the database depending on a given attribute, but for this optimization, *Nonclustered* indexes were used, as there is no need to sort the data.

2.2 Front-end

The client side has been developed with Angular[8][9], HTML, CSS[10] and with the help of some libraries for the stylizing, such as Angular Material[11][12] and Creative Tim.

Angular

Angular is a TypeScript-based[13] open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

Angular 8 was the version used for implementing this application's front-end or client side. This particular version features differential loading for all application code, dynamic imports for lazy routes, web workers, TypeScript 3.4 support, and Angular Ivy as an opt-in preview. Angular Ivy opt-in preview includes:

- Generated code that is easier to read and debug at runtime
- Faster re-build time
- Improved payload size
- Improved template type checking
- Backwards compatibility

Some advantages found on Angular were its compatibility with .NET, its code re-usability and adaptation to multiple platforms, its speed, performance and facilities to internationalize the application and provide responsiveness.

Chapter 3
Methodology

In this chapter the methodology followed for this development will be explained, along with the planning and monitoring that were followed to put it in practice.

3.1 Scrum

The methodology used was **Scrum**[14], adapted to a single person environment. Scrum consists of an agile methodology[15] based on *sprints*, implying various roles, and it is focused on change adaptation. Scrum affirms that the problem to solve cannot be totally defined, as it will vary as time passes and the client may demand changes, so it is focused on change adaptation through the sprints, which will develop increments on the application. Scrum's goal is to allow fast adaptation to changes, and to maximize the productivity of the development team.

3.1.1 Roles

The Scrum methodology defines a series of roles, from which we can highlight the following:

- **Product Owner:** The Product Owner represents the stakeholders of the product and their needs, and they are responsible for defining a solution useful for them, defining and prioritizing the **User Stories** that will conform it. It is crucial for this role to be able to communicate between team members and stakeholders. In this particular case, as a single person adaptation of Scrum, this complexity was avoided.

- **Development team:** They are a self organized team, usually between 3 and 9 people, responsible of developing the increments of the solution in each sprint, which should be deliverable. They don't have specific roles among them, they work as a group with the potential to develop the project. Adapting this role to a single person did not suppose

further complexity rather than the need for this person to acquire knowledge of a full-Stack developer.

- **Scrum Master:** Is responsible of ensuring that the Scrum process is followed correctly, avoiding distractions from the development team and helping them cooperate. Their task is to improve productivity and achieve the solution objectives.

For this project, adapted to a single person environment, every role is executed by the same person, but this methodology is still useful helping the developer to keep in mind it's various ways of thinking about the solution, for example, as the Product Owner when they need to select User Stories for the Product Backlog, or the Development team when programming or estimating tasks in the Sprint Planning.

3.1.2 Artifacts

Scrum Artifacts are those elements which represent work or value to provide the solution, and its continuous inspection and change adaptation. They must be designed using business language in order to allow everyone (including stakeholders) to understand every artifact.

Product Backlog:

The User Stories that will conform the solution are defined in the Product Backlog. Whenever new changes arise during the development, they must be added to the Product Backlog, keeping it updated with every feature that will conform the solution. This backlog will be reviewed at the beginning of every sprint to define the new User Stories to implement. It must also keep track of which User Stories are finished, which are in progress and which have not been started yet. For this, the tasks which define a User Story must have a definition of what it means to be completed, which is known as an acceptance criteria.

For this project, the Product Backlog was represented as a list of User Stories, which initially consisted on searching, comparing and viewing details of events and artists. These User Stories grew during the development as new functionalities which added value to the product were found. This Backlog kept the state of the User Stories updated (new, active, closed...) and provided a global view of the state of the application and which tasks remained to test or to implement.

Sprint Backlog:

The User Stories from the Product Backlog that will be implemented in a specific sprint are the ones listed in the Sprint Backlog, along with a planning for the increment development and delivery. When every sprint begins, the team defines this backlog at the Sprint Planning,

choosing which User Stories will be implemented as an increment of the complete solution, and estimates the remaining work. When time passes and tasks advance, this backlog updates the remaining work estimation.

The Sprint Backlog worked similarly as the Product Backlog for this project, as a list of User Stories to implement and their state. It helped keep track of tasks to implement at every point in the development without getting lost in new functionalities or additions, which were added to the Product Backlog for later implementation. This was particularly useful, as the student usually had new ideas for additions to the application, and easily lost focus on tasks of the current sprint.

Increment:

It's the sum of every element at the Product Backlog completed during a sprint. This increment must always be an usable solution, even if it's not going to be released.

In this project, the increments where exponentially bigger each sprint, given the development team had more experience within time, and more complex functionalities were provided on every increment.

3.1.3 User Stories

They are work elements and requisites, defined in business language so that everyone can understand them. The **Product Owner** is the responsible of their redaction and prioritization, and they are the basic unit of every sprint. The User Stories defined for this project are covered in detail on section 4.4 (page 18).

User Stories are inter-dependent, negotiable, estimable, and have *acceptance criteria* associated to determine whether they are finished. They are concise, but a User Story has to be granulated into smaller tasks to simplify the work to perform as much as possible, and to be able to make a homogeneous distribution of tasks among the development team.

3.1.4 Sprints

A sprint is a time lapse chosen by the team, usually 15 days or two weeks, in which an increment of the solution is developed. It's duration must be constant during the project, and at the end of every Sprint, the next one begins. For this project, Sprints have a fixed duration of **2 weeks**. They are covered in depth on section 7.1 (page 43).

The different reunions of the Sprint help plan, implement and review the work performed during it. This allows the continuous followup of the project and eases change adaptation and the inclusion of new requisites.

- **Sprint Planning:** It takes place before every sprint, in order to select from the Product Backlog which User Stories will be implemented in said sprint. This stories are usually ordered depending on their priority. It's crucial to estimate the complexity of every User Story in order to organize the work to complete during the sprint. Once the User Stories are selected and estimated, they are divided in smaller tasks to ease the assignment of the work to perform.

- **Daily Meeting** or Daily Scrum: It's a short meeting, usually under 15 minutes, which takes place every day, which focuses on stating what everyone in the team did the previous day, what they are going to do today and with which obstacles might need help with. This helps the team to keep track of what everyone is doing and avoids the need of other bigger reunions. Even in this single person environment, the Daily Meeting proved to be useful to help the student reorganize her work and focus every day.

- **Sprint Review:** Is the reunion after the sprint takes place, in order to analyze if the work was completed successfully, or if there were tasks or features that could not be completed and why. The Product Owner and the stakeholders will decide if the sprint objectives are accomplished, and if they are not, they will be delayed to be implemented or corrected in the next sprint.

- **Sprint Retrospective:** It takes place before a new sprint begins, to present any problems that raised in the development and correct them for the next sprint, and also to enhance the practices that worked well and to encourage the team to keep using those advantages. It must not be confused with the Sprint Review, which focuses on the User Stories implemented in the previous sprint, while the retrospective focuses on how the work was performed, problems in team coordination, or better practices for the next sprint.

Some advantages of this methodology were:

- It helped the development team keep track of every defined User Story and what to implement in every sprint, which helped the student avoid focusing on new functionalities rather than the current sprint tasks.

- High coordination capability, as in a single person environment, Scrum does not have the inconvenience of the difficulty for organizing the team members. It was easier to keep the backlogs updated and select new tasks to perform when previous ones were finished, as the student knows at every time, which tasks are in progress, finished, etc.

- It made the student think as a Product Owner in order to find new features during the development, defining and adding them to the Product Backlog.
- Flexibility and change adaptation to these new functionalities allowed the student to create a bigger solution than the one initially proposed.
- It reduced the excess of documentation of classical methodologies, which would be unnecessary in a single person environment, and excessive for this kind of application.

Chapter 4

Analysis

In this chapter, the project to develop and its conceptual model are described, along with the architecture to be implemented for this project. Requirements for the system will also be covered, as they are tightly related with the architectural decisions.

The requirement analysis is usually performed at an early development stage, being a fundamental part of software development. In this phase it's necessary to determine requirements and conditions the software has to satisfy to result in a viable solution for the user's needs. It's vital for a good development to deeply analyze the system's scope and it's functionalities before the design begins.

4.1 General description

This application will allow users to search music events using advanced filtering criteria in order to compare and choose between events, keep updated of artists attending them, tickets prices, location and date of the events. For this purpose, an advanced search will be provided to every user without registering, but for following content, a registered account will be required. Registered users will also have access to custom notifications on changes from music events and artists they follow, such as, new tour dates from an artist or ticket availability from a music event.

To administrate the application's content, artist users will manage artists profiles and event administrators will manage various events. This will allow updating whether an artist is attending to a specific event, their genres, the tickets available for an event and their prices, etc. These changes can be notified to users with custom notifications in which administrators will be able to select the content they want to announce, along with a description.

A high level representation of these requirements is provided in figure 4.1.

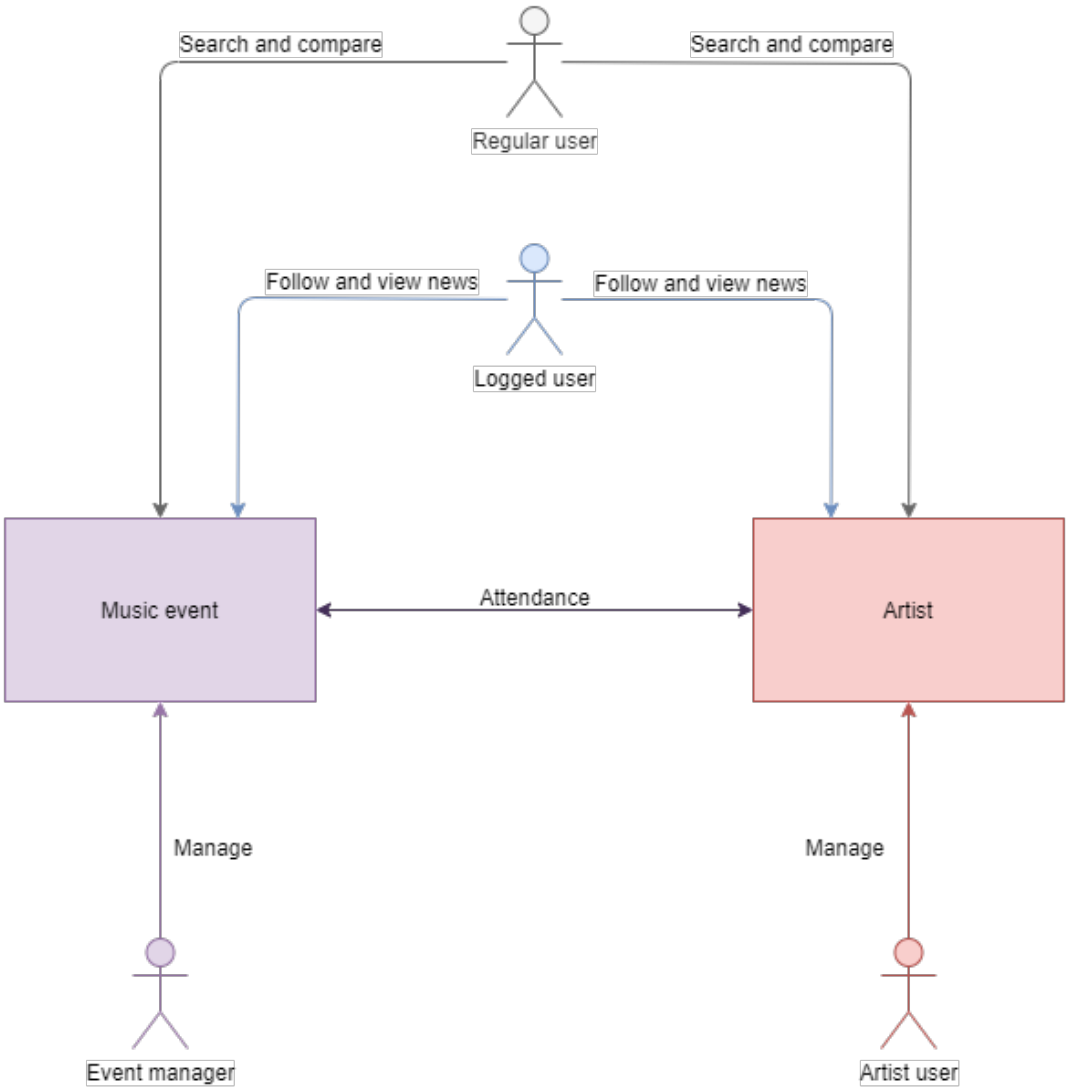


Figure 4.1: General representation of requirements by actor type

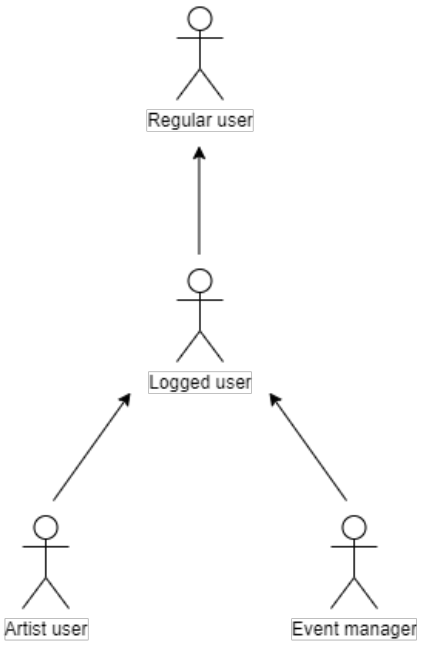


Figure 4.2: Actors diagram

4.2 Actors

The intended users of the application will be anyone who seeks information about artists and events, but registered users will have access to extra features such as following artists and events of its interests to keep updated about them. For managing this information provided to the regular users, there will be two kinds of administrator users, which will have access to more advanced features. These administrator user types are the **Artist users**, which will administrate one artist profile, it's associated information and events they'll attend to, and the **Event managers**, which will be able to administrate various events, update their confirmed artists, dates, tickets and all associated information. Actors for this application are represented in figure 4.2.

Regular user

This user is the most simple one, having access only to the features provided to non-registered users. This features are searching events and artists with all filtering criteria, and accessing to some general news for the most popular events and artists of the application.

Logged user

This user is an expansion of the previous one, adding to it's functionalities, the possibility to follow events, artists and genres to keep track of their updates, new tour dates, and so on. In addition, they will have in the dashboard page, links to all events and artists they follows, along with the possibility of accessing every profile individually. For this users, the general news page will be customized with specific news of the events, artists and genres they follow.

Artist user

This user will administrate an artist profile, providing general information about a band, singer or musician such as name, description, location, genres and social media. It will also manage which events they'll attend to, and provide custom notifications for their followers about this tour dates. In case a tour date is canceled, the associated notifications about it will be automatically deleted, and the artist will optionally publish a cancellation notification, allowing them to describe the reason for the cancellation.

Event manager

This user will administrate various events, providing information about them such as dates, location, description, artists attendance and available tickets and their prices. This events don't usually post all confirmed artists at once. Instead, they will provide news on more confirmed artists along time. To notify this, event managers can publish a notification on the confirmed artists they choose, with a custom description. The same happens for ticket availability. In case of cancellations, the event administrator will be able to notify them in the same way as the artists, with a custom notification description.

4.3 Conceptual model

The entities that were found to model this application come from the music industry, mainly from music events and artists, to provide the user useful information about them. An in-depth definition of this application's entities and attributes can be found on section 5.2.1 (page 26). The Entity Relationship Diagram (Figure 5.2, page 27) and the Entity Data Model (Figure 5.3, page 28) show graphical representations of these entities and their relationships.

4.4 User Stories

As described in the Methodology section 3.1.3, User Stories are work elements which define the application's requirements. Analyzing the application, four main groups of User Sto-

ries where found; searching artists and events, user administration, artist administration and event administration. These User Stories conform the Product Backlog, which was updated during the development to include changes or details of the features.

4.4.1 Search

Both types of searches will be available for every user of the application, without a registered account.

Music event search

As a **User** of the application, *I want to* be able to search and compare events using advanced criteria *so that* I find the most suitable ones according to my personal taste.

To allow the user music event comparison, the following criteria will be provided:

- Event name (partially or completely including a given text, case insensitive).
- Begin and end date of the event within a provided date range.
- Price of the event tickets (for events with available tickets).
- Location of the event, by country and city.
- Main genre of the artists involved in the event.
- Artists attending to the event with two variants; all artists included, or some of them.

All this criteria is optional, allowing the user to simply search all the events on the application. Once the user has introduced the filtering criteria they want to consider, they will be able to search and view a list of events matching this criteria, ordered by most criteria coincidence. This list will provide, for each event, a link to the event details and a button to follow it in order to get notified of future updates.

Artist search

As a **User** of the application, *I want to* be able to search and compare artists using advanced criteria *so that* I find the most interesting ones according to my personal taste.

To allow the user to search for specific artists, the following criteria will be provided:

- Artist name (partially or completely including a given text, case insensitive).
- Artist genre (artists might have many genres, so it is sufficient with one coincidence).
- Artist location, understood as the country and city of origin of the artist.

All this criteria is optional, allowing the user to simply search all the artists on the application. Once the user has introduced the filtering criteria they want to consider, they will be able to search for a list of artists matching it and a link to their artist profile and follow button, to get notified of future updates.

4.4.2 User administration

These User Stories involve functionalities requiring a personal account, and its creation.

Create account

As a **User** of the application, *I want to* be able to register a personal account *so that* I can follow artists and events and view personalized notifications.

Log in

As a **Registered User** of the application, *I want to* be able to log in it *so that* I can access functionalities regarding the logged user requirement.

Log out

As a **Registered User** logged in the application, *I want to* be able to log out of it *so that* I can stop allowing access to my account's features on a particular device.

Configure profile

As a **Registered User** logged in the application, *I want to* configure my profile (name, login name, email, city, country, language...) *so that* I can log in the application, maintain my personal information updated, receive appropriated notifications depending on my location and customize the application's language.

Delete profile

As a **Registered User** logged in the application, *I want to* be able to delete my personal account *so that* I can remove all associated data about myself from the application.

Follow event

As a **Registered User** logged in the application, *I want to* be able to follow events *so that* I stay updated on any news they publish, such as, new confirmed artists or ticket prices available.

Unfollow event

As a **Registered User** logged in the application, *I want to* be able to stop following events *so that* I no longer receive notifications about them.

Follow artist

As a **Registered User** logged in the application, *I want to* be able to follow artists *so that* I stay updated on any news they publish, such as, new tour dates.

Unfollow artist

As a **Registered User** logged in the application, *I want to* be able to stop following artists *so that* I no longer receive notifications about them.

Follow genre

As a **Registered User** logged in the application, *I want to* be able to follow particular music genre *so that* I receive the most popular news of events or bands involving it. This is interesting in order to discover new bands or concerts involving a genre I like in case I don't know many artists of that said genre.

Unfollow genre

As a **Registered User** logged in the application, *I want to* be able to stop following genres *so that* I no longer receive notifications about artists or events involving it.

4.4.3 Artist administration

These User Stories will only be available for accounts registered as artists, as they are provided for creating and customizing an artist profile.

Configure profile

As an **Artist user** of the application, *I want to* configure my profile details (name, description, list of genres, band location, social media...) *so that* all criteria for the users to find me is available, as well as my artist profile details.

View tour dates

As an **Artist user** of the application, *I want to* view all my confirmed tour dates *so that* I can properly manage them; publish notifications about them or remove tour dates.

Add tour date

As an **Artist user** of the application, *I want to* be able to confirm my attendance to certain events, searching them by name *so that* they are shown in my profile's events list, and users searching events including me find that event.

Notify tour date

As an **Artist user** of the application, *I want to* create a notification for a confirmed tour date, with a custom description, the date of the confirmation and a link to the confirmed event, *so that* I explicitly show my followers I'm attending to that event. This notifications will also appear on my profile, along with the list of events I'll attend to.

Cancel tour date

As an **Artist user** of the application, *I want to* be able to cancel a tour date which is no longer taking place *so that* it does no longer appear on my profile and all associated notifications about it are automatically removed.

Optionally, *I want to* be able to publish a cancel notification with a custom description explaining why the tour date was cancelled, in order to explicitly show my followers this event is no longer in my tour dates.

Create custom notification

As an **Artist user** of the application, *I want to* be able to post a notification *so that* I can show my followers any interesting data about me.

Remove notification

As an **Artist user** of the application, *I want to* be able to remove any notification I published *so that* it's not showed on my profile anymore (This process will be automatic in case of a tour date cancellation, as explained before).

4.4.4 Event administration

These User Stories will only be available for accounts registered as event administrators, as they are provided for managing one or more events.

Create event

As an **Event administrator** of the application, *I want to* be able to create a new event *so that* I can manage it.

View managed events

As an **Event administrator** of the application, *I want to* view the events I created, listed in an administrated events page, *so that* I can properly manage them.

Configure event

As an **Event administrator** of the application, *I want to* be able to configure my managed events data, setting their start and end date, location (country, city and address), description, social media (such as official web page, tickets sale points, Instagram, Twitter and Facebook accounts) and all associated criteria, *so that* these events are kept updated for users to find them.

Confirm artists

As an **Event administrator** of the application, *I want to* confirm artist attendance to every managed event *so that* my events are interesting for users looking for those artists, show the attending artists on the event profile and allow searches for events including those artist find my event. To confirm artists attending an event, *I want to* find them by name and select the ones I want to confirm, which will be added to the list.

View confirmed artists

As an **Event administrator** of the application, *I want to* view, for each of my managed events, a list of attending artists, *so that* I can properly manage them; publish notifications about them or remove any of them.

Notify confirmed artists

As an **Event administrator** of the application, *I want to* be able to select some of the confirmed artists attending an event and create a notification with a custom description *so that* followers of the event are notified about them and this notification appears on their news feed and on the event's page, along with the confirmed artists list.

Remove confirmed artist

As an **Event administrator** of the application, *I want to* be able to cancel an artist attendance to an event, *so that* it does no longer appear as attendant of my event, and to automatically remove it from any previous notification about their attendance (this does not affect other artists confirmed in the same notification, but in case the notification only included that artist, it will be removed completely).

Optionally, *I want to* be able to publish a notification about the cancellation with a custom description *so that* I explicitly show my followers this artist is no longer attending that event and why.

Create ticket

As an **Event administrator** of the application, *I want to* create available tickets for every managed event *so that* users know every option to access the event, show the available tickets on the event profile and allow searches for events with available tickets under a certain price to find my event.

View available tickets

As an **Event administrator** of the application, *I want to* view, for each of my managed events, a list of available tickets, *so that* I can properly manage them; publish notifications about them, edit or remove any of them.

Notify available tickets

As an **Event administrator** of the application, *I want to* be able to select some of the tickets available for an event and create a notification with a custom description *so that* followers of the event are notified about them and this notification appears on their news feed and on the event's page, along with the available ticket list.

Remove available ticket

As an **Event administrator** of the application, *I want to* be able to remove a ticket of an event, *so that* it does no longer appear as available. This does not remove notifications about it. Instead, it is set as "Sold out".

Create custom notification

As an **Event administrator** of the application, *I want to* be able to post a notification *so that* I can show my followers any interesting data about my events.

Remove notification

As an **Event administrator** of the application, *I want to* be able to remove any notification I published in any of my managed events, *so that* it's not showed on the events profile anymore (This process will be automatic in case of an artist cancellation, as explained before).

Chapter 5

Design

In this chapter, design and architectural decision for this application will be covered.

5.1 Architecture

This project followed a Client-Server architecture[16][17], having a server-side or back-end and a client-side or front-end. The client-side follows the MVC architecture, while server-side is structured as a layer architecture, having a model and a web layer. The global architecture of the application is represented in figure 5.1.

5.2 Server side

The server-side of this application was developed using .NET framework and it's divided in two main layers; the web layer, which consists of the web APIs offered to the client-side, and the model layer, which provides the domain of the application, the business logic and the data access.

5.2.1 Model layer

The Model layer contains three sub-layers: the **Domain layer**, which contains the persistent entities of this application, the **Business logic layer**, which contains the services and the **Data access layer**, which contains Data Access Objects to retrieve information from the database. To understand how to implement the application's logic, we need to design the entities and their relationships properly. To obtain data from the database in order to work with these entities, the DAO pattern will be followed, and to isolate the business logic, we will use services.

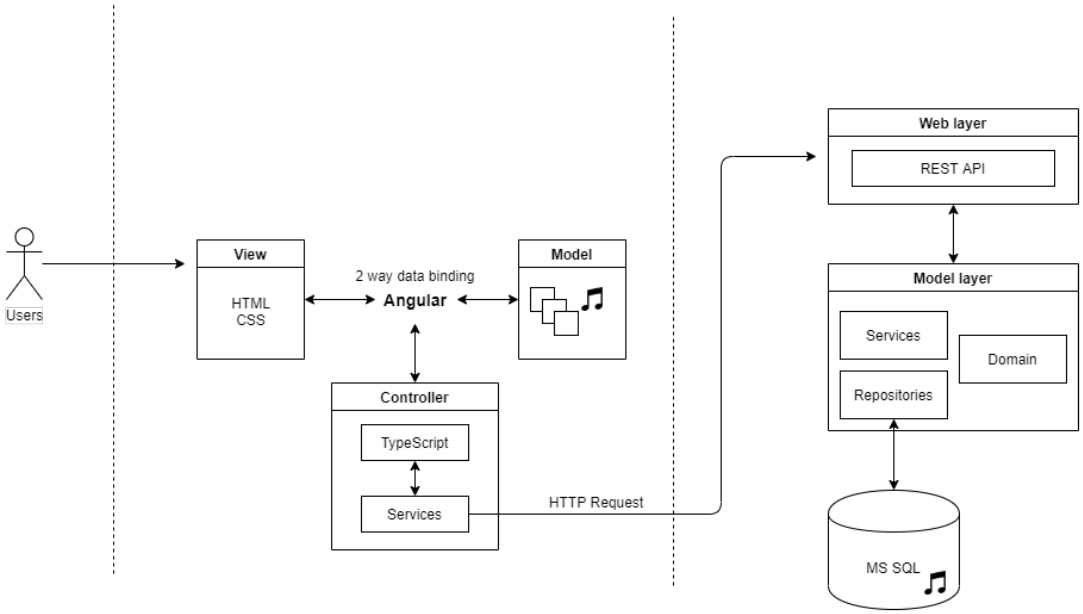


Figure 5.1: Global architecture diagram

Domain layer

The **persistent entities** for this application represented in the Entity Data Model (figure 5.3) and the Entity Relationship Diagram (Figure 5.2), come from the music industry, mainly form music events and artists, to provide the user useful information about them. They will be covered explaining their attributes and relationships (trivial attributes such as Id or Name are not explained).

Entities

User A registered user account. It identifies a unique user of the application.

- Id
- Login name: Name the user will use to log in the application. It has to be unique.
- User name: Name which will be used to refer to this user.
- Email
- Password
- City id: 1-N relationship to set the users city.
- Country id: 1-N relationship to set the users country.

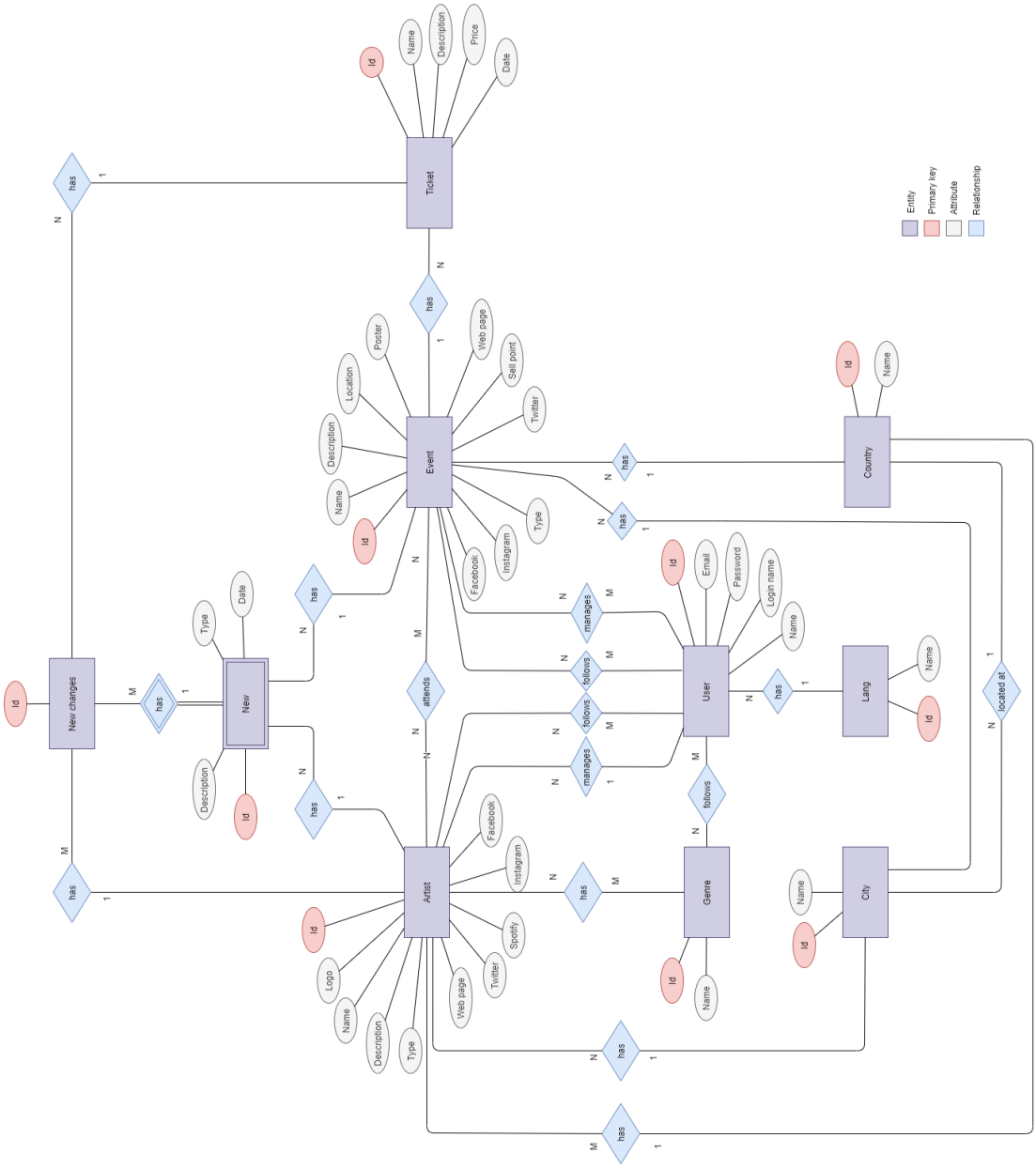


Figure 5.2: Entity relationship diagram

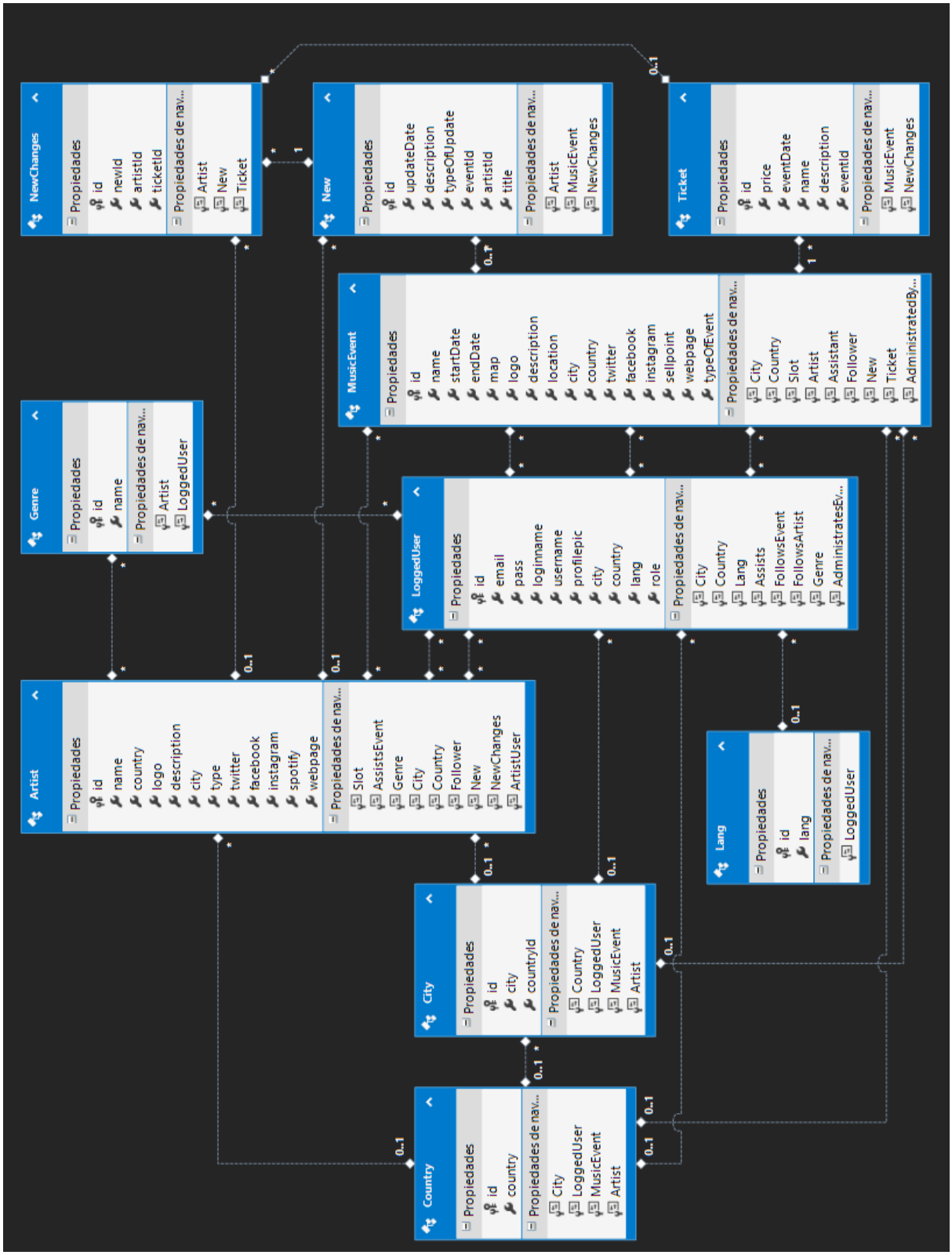


Figure 5.3: Entity data model

- Language id: 1-N relationship to set the users language.
- Role: Sets the user's role, in order to guarantee only certain users access some functionalities.
- Administrated events: 1-N relation with the events this user can manage, if they are an Event manager.

Music event A music event (concert, festival...) users may find in the application and view all related data.

- Id
- Name
- Start date
- End date
- Description
- City id: 1-N relationship to indicate the city where this event takes place.
- Country id: 1-N relationship to indicate the country where this event takes place.
- Location: Address of the event's location.
- Poster: Image to show bands attending the event and their custom logo
- Type of event: Festival or concert.
- Web page: Link to the official web page of the event.
- Sell point: Link to the official ticket sell point of the event.
- Instagram: Link to the event's Instagram page.
- Twitter: Link to the event's Twitter page.
- Facebook: Link to the event's Facebook page.
- Artist ids: N-M relation with artists attending this music event.

Artist An artist (band, singer, guitarist...) users may find in the application and view all related data.

- Id

- Name
- Description
- City id: 1-N relationship to indicate the city where this artist is from.
- Country id: 1-N relationship to indicate the country where this artist is from.
- Logo: Image to represent this band
- Type of artist: Band, singer, guitarist, bassist…
- Web page: Link to the official web page of the artist.
- Instagram: Link to the artist's Instagram page.
- Twitter: Link to the artist's Twitter page.
- Facebook: Link to the artist's Facebook page.
- Spotify: Link to the artist's Spotify page.
- User id: 1-1 relation with the user account a registered artist uses to log in the application and manage their profile.
- Genre ids: N-M relation with music genres this artist has.

Ticket An available ticket associated to an event users may find in the application. It includes it's price, in order to find events having available tickets under a given price.

- Id
- Name
- Description
- Price
- Date: Date of the event which this ticket allows access (if it grants access to the whole event, it's null).
- Music event id: 1-1 relation with the music event this ticket belongs to.

Genre A music genre an artist can have. It's also used to identify the music genres of an event, depending on the most common genre of artists attending it.

- Id

- Name

New Notification for some data changed for an artist or event, such as, new tour date confirmed by an artist or available tickets confirmed for a specific event. If published by an artist, it will be identified by the *artist id* and the *event id* will identify a tour date, while if it corresponds to an event, it will be identified by the *event id* while the artist id will be set to null (to access a list of confirmed artists, the New changes entity will be used).

- Id
- Title
- Description: Text introduced by the artist or event manager to explain whatever they want about the notification.
- Type of new: To identify the type of new (Tour date, artist confirmation, cancellation...)
- Date: Date and time at which this notification was published.
- Artist id: 1-N To identify the artist who published this notification, if published by an Artist user.
- Event id: 1-N To identify to which event this notification belongs to.

New changes Represents the relationship between a notification (New) and some changes added to it, for example, when confirming artists attending to a specific event, to associate those artists with the notification.

- Id
- New id: 1-N To relate it to the new whose changes are stored in this entity.
- Artist id: 1-N To relate the new with the artists notified on it, if any.
- Ticket id: 1-N To relate the new with the tickets notified on it, if any.

Country Representation of a country in order to find events depending on their location, to set the user's location to provide notifications of events near them or to set the location of an artist.

- Id
- Name

City Representation of a city in order to find events depending on their location, to set the user's location to provide notifications of events near them or to set the location of an artist. They will depend on a country.

- Id
- Name
- Country id: 1-1 To represent in which country this city is.

Language Representation of the possible languages to select the preferred one for the application.

- Id
- Name

Data access layer

This layer used the **DAO** or Data Access Object pattern to access the Database, providing CRUD methods for our entities and isolating those methods from the rest of the application. For this application, the following DAOs where created: *MusicEventDao* for every music event related data, *ArtistDao* for artist data, *UserDao* for users and administrators, *GenreDao* for music genres, *NewDao* for news from the events and artists, *CityDao* for cities, *CountryDao* for countries and *LangDao* for language.

In many cases, the data requested by the client-side will not match exactly the persistent entities format, or sending those entities might even be dangerous because they expose some attributes the client-side should not be able to see. For those cases, and to reduce excess of information sent to the client, **DTOs** or Data Transfer Objects are used, creating specific objects to send the front-end the appropriate data, hiding unnecessary attributes and/or adding others which provide extra information, such as, follower count on artists or events.

Business logic layer

Services are used to isolate the business logic, and their design for this application followed a similar approach as DAOs: *MusicEventService* for every music event related logic, *ArtistService* for artist logic, *UserService* for users and administrators, *GenreService* for music genres, *NewService* for news from the events and artists and *LocationDao* for cities, countries and language. These services are represented in figure 5.4.

5.2.2 Web layer

The web layer is composed by the web REST APIs provided to the client-side. This layer is the higher one in the server-side, attending the client's petitions and providing responses using the model layer to obtain them.

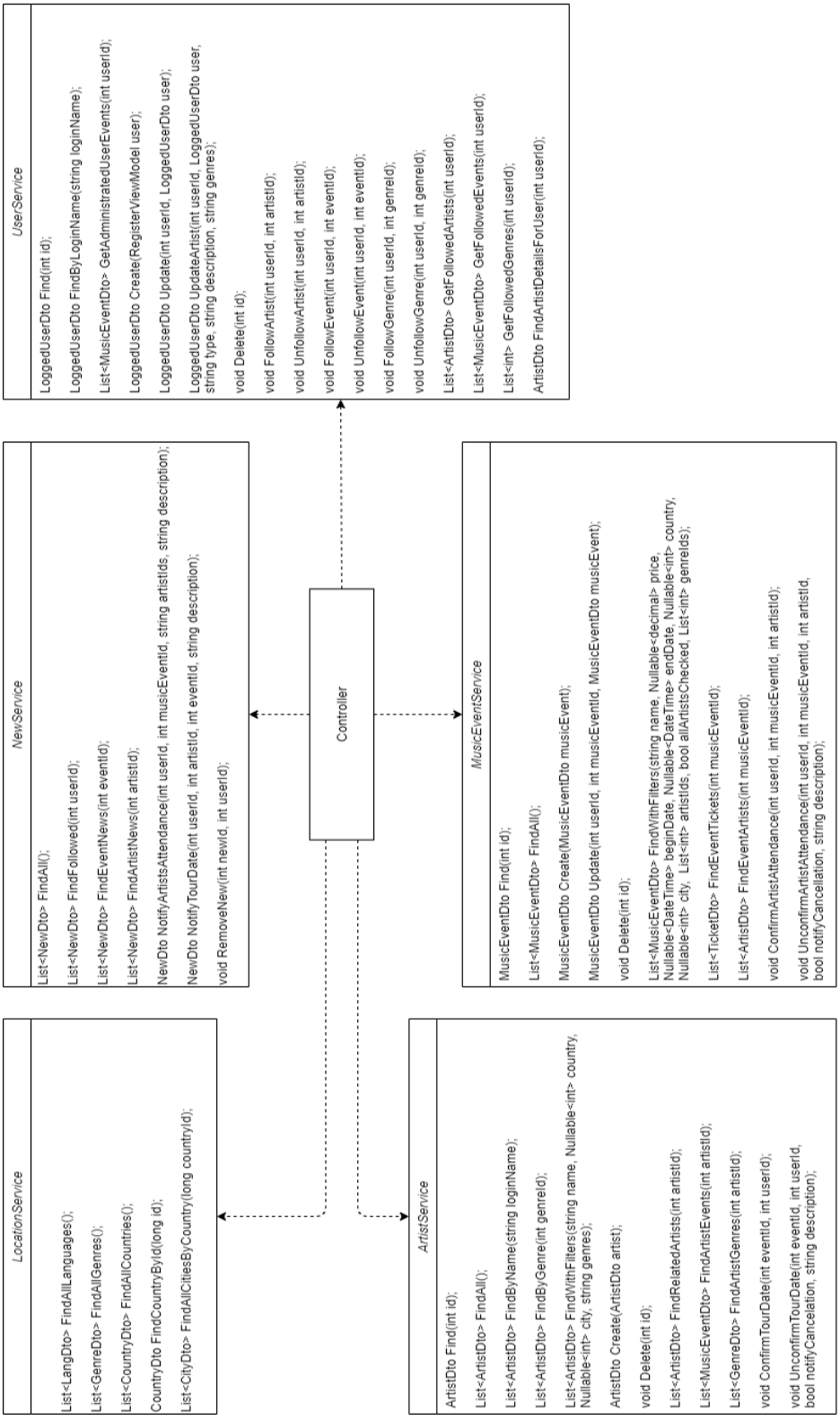


Figure 5.4: Services diagram

The communication between the client-side and the server-side is performed following the REST architecture, which allows this distribution. REST uses HTTP calls on specific routes to obtain data. Using a different call on the same route would trigger different server-side methods, allowing to distinguish different purposes.

- **GET:** It's used for reading one or a collection of resources from the application. It's use on a collection of resources might use a route as "MusicEvent/" while a petition for a particular resource would use a route like "MusicEvent/{music event id}".

- **POST:** Sends a representation of a resource to the application, usually to create one. Following the previous example, to create a music event, the route "MusicEvent/" would be used, and due to the different type of call, the server-side distinguishes it from an attempt to get all music events calling to that same route with a GET call.

- **PUT:** It's used to update a resource, so it sends an updated representation of an existing resource of the application. It is applied to a particular resource, through routes like "MusicEvent/{music event id}".

- **DELETE:** It's used to delete an existing resource, sending only it's identification, so a route like "MusicEvent/{music event id}" would be enough.

5.3 Client side

The client side of this application was developed using Angular and followed a Model-View-Controller or MVC architecture[18]. This design pattern consists of three interconnected elements and aims to separate internal representations of information from the ways information is presented to and accepted from the user. In order to do this, the *view* will provide the user with the visible interface, the *controller* will receive the user's petitions and work with the *model* for managing the data of the application. These interactions are represented in figure 5.5.

5.3.1 Model

The model is responsible for managing the data, logic and rules of the application. It consists of the representation of said data, and it receives user input from the controller. In order to send data to the server-side, the web service will provide data through HTTP in a JSON format.

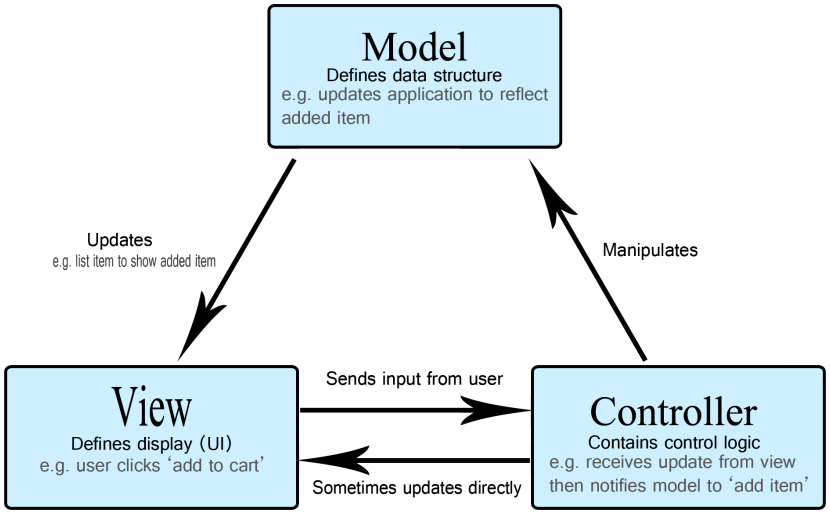


Figure 5.5: Model View Controller diagram

5.3.2 View

The view presents the model in a particular format, providing the user some information the controller has prepared. It is composed by the HTML and CSS files that conform the interfaces of the application.

5.3.3 Controller

The controller responds to the user input and performs interactions on the data model objects, calling the model's methods to obtain necessary data for the view.

5.4 Interface design

The design for this application was intended for every kind of user who wants to find information on music events and artists, so it's accessibility is primordial to allow it's use to the biggest audience possible. For this purpose, the interfaces were deeply polished in order to allow responsiveness in any device and to aid users with disabilities to be able to properly use them.

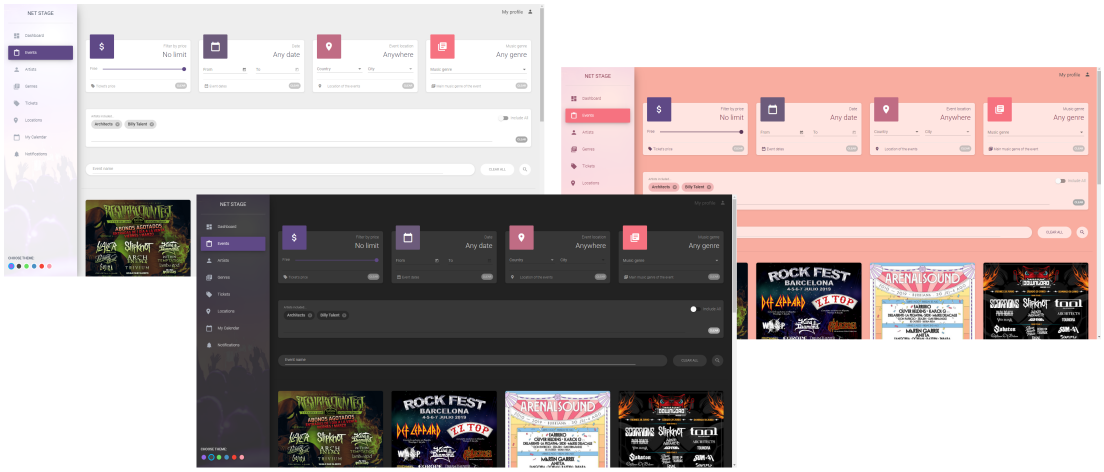
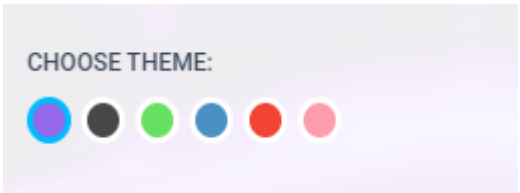


Figure 5.6: Example of available themes

Accessibility and themes

The application allows accessing form fields through keyboard and provides feedback on required or incorrect fields in forms in addition to back-end validation, which is always necessary for safety reasons. It does also provide tooltips on management functionalities for administrator users.

As an addition to the main theme of the application, more themes with different color palettes are available for the users to choose from. This allows to select a *dark mode*, which will be more suitable at night to avoid too light backgrounds, a *colorblind mode* which provides a color palette suitable for colorblindness, as seen in figure 5.7, as it chooses colors distinguishable in the three types of colorblindness, and some other themes to provide a customized interface. An example of these themes is shown in figure 5.6.



Responsiveness

A responsive application is an application which adapts to different screen sizes, arranging the interface in different layouts and adapting it in order to be used in different devices.

The application is responsive for any device size (large computer screen, medium or laptop screen, tablet or phone), adapting the lists of events or artists to the screen width by reducing the number of columns displayed as the screen gets smaller. In a mobile phone,

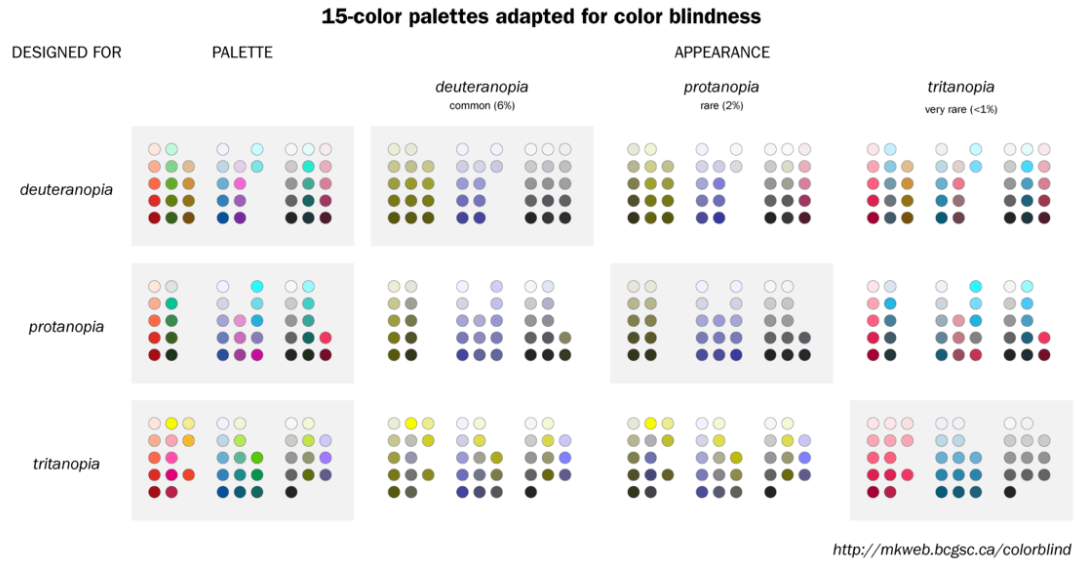


Figure 5.7: Color palettes adapted for color blindness

only one column would be displayed and as the screen gets bigger, two to five columns may be displayed. Other kind of adaptations were made to ease it's use in small screens, such as, hiding the menu and adding a hamburger button to show it. The search event page is used as an example of how responsiveness is implemented in this application in figure 5.8.

Internationalization

The internationalization of a software application consists in providing the necessary structure in order to allow language changes without changing code.

For the internationalization of interface texts, the *Translate* Angular library was used, allowing texts in every view page to be set depending on the user's selected language. This library uses JSON files in order to store every language version of the texts included in the application, and selects the appropriated translation depending on the user's language choice.

For text stored on the database, such as notification descriptions written by administrators, there is no affordable solution without interfering with the administrators freedom to post customized texts, but a possible future solution will be to provide an embedded translator such as Google Translate on every custom post, artist or event description, like in other social media.

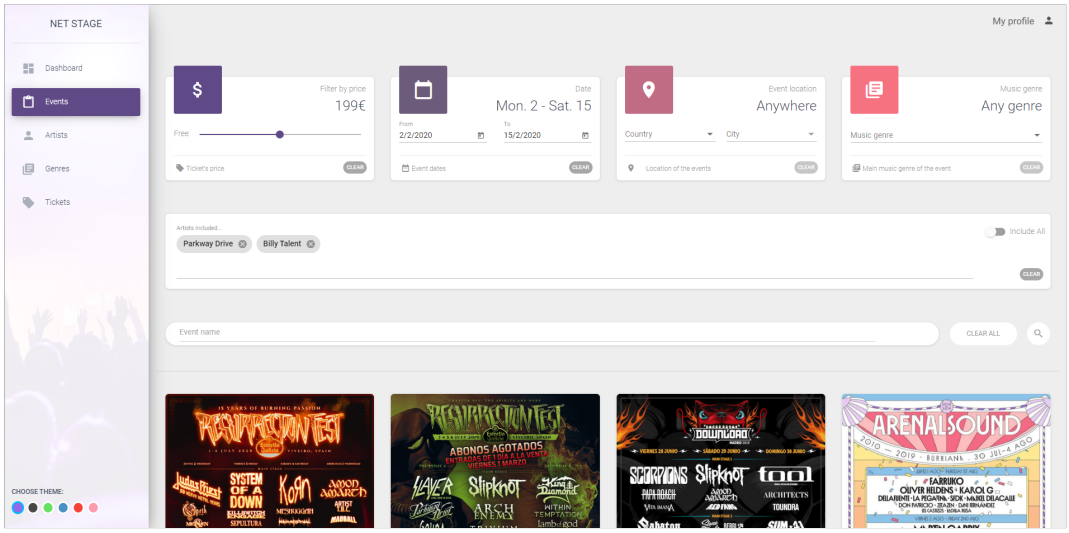


Figure 5.8: Example of responsiveness

Chapter 6
Planning and monitoring

In this chapter the planning for the development of the application will be explained, along with initial budget and final results.

6.1 Planning

This section will explain the initial planning, before the implementation began, but as an agile development was followed, it is necessary to plan every sprint instead of planning of the entire application initially. This allows change adaptation as explained before, in chapter Methodology (3).

As this development followed Scrum adapted to a single person environment, the tasks could not be developed in parallel. This supposes a work overload that the planning will take into account. As explained in section 3.1, Scrum defines the application to implement through the *Product Backlog*, and divides work in sprints which will deliver a usable version of the product. This project defined 2 week sprints and a 6 hour working day, being 6 hours equal to one *Story Point*. Story Points are a way to measure time using the rounded Fibonacci scale, which avoids wasting time estimating tasks. Each task of a sprint is estimated by the development team before the sprint begins, in order to select which tasks can be implemented during it.

The project will be developed in **6 sprints** (2 weeks each), having a total duration of 3 months.

Sprint: 2 weeks x 5 days x 6 hours = 60 hours (30 hours per week).

Total hours: 6 Sprints x 60 hours per sprint = 360 hours.

6.1.1 Budget

To calculate the project's budget (Table 6.1), it is necessary to take into account not only workers payments but also tools and licenses. This project is developed in a personal laptop,

Table 6.1: Final budget

Resource	Cost	Work time	Total
Software engineer	20€/h	360h	7,200€
Computer equipment	800€	3 months	50€
Licenses and server	0€	-	0€
			7,250€

using some free open source licenses and tools, and also licenses provided for free to UDC students, such as, Visual Studio Enterprise 2015. To deploy the project, a free server will be used. The price of the personal laptop used to develop this project was 800€, and it was used during a period of 4 months (from the start of the analysis phase).

The single person working as software engineer, analyst and designer:
20€/h x 360 hours = 7,200€

Personal laptop amortization:
(800€/(4 years x 12 months)) x 3 months = 50€

Web server: 0€

Licenses: 0€

6.2 Monitoring and followup

As explained in the Methodology chapter 3, the planning did not take place only before starting the implementation, but through all the process, at the beginning of every sprint. This allowed change adaptation and avoided loosing track on which work to focus during the development. During the development of the application, there also raised new features that could provide useful functionalities to the user, which were also included in the Product Backlog and implemented in future sprints, without loosing track of what was being implemented at the current one.

These Sprints along with the Product Backlog also allowed to monitor the state of the application, ensuring at every point in the development, which work was completed and tested, which was being implemented or had been delayed and which new features were added for future implementation.

6.2.1 Sprint planning

Each sprint began with a *Sprint planning* in which elements from the Product Backlog were selected to be developed during it. This meeting helps defining and estimating tasks and distribute work that will be performed. For each sprint, the planning and the elements that conform the Sprint Backlog are the following:

Sprint 1

The main goal of this sprint was starting up the skeleton of the application, creating the database with the initial entities that were needed (some did not have all attributes yet), and get familiarized with Angular and the development environment.

The User Stories selected for this initial Sprint were the basic operations from User administration: Create account, Configure profile (which includes viewing it), Delete profile and logging in and out of the application. 4.4.2

Sprint 2

This Sprint's focus was setting the structure for searching and filtering evens and artists, allowing it's comparison. The User Stories implemented at this point where the Searches 4.4.1. The basic interfaces for the application where constructed, setting an initial layout for the event and artist list view and their filters.

Sprint 3

After searching events and artists, the user's next need was clearly following them in order to have faster access to their interests without searching every time. At this point, the follow/unfollow and view details options were added to both events and artists, to provide this functionalities. The layout for details page for events and artists provided more information than the previous search list, and included lists for events a particular artist would attend, artists a particular event would include (reusing the previous list component from the search page), and a new available ticket list for an event.

Sprint 4

To allow artists to configure their own profile and manage their tour dates, the User Stories from Artist administration were selected for this Sprint 4.4.3. This provided some useful new functionalities and data for Artists, such as, social media, which was not included before in the Artist profile attributes.

Sprint 5

To allow event managers to configure their own music events and manage attending artists, create and update available tickets and all related information about the events, the User Stories from Event administration were selected for this Sprint 4.4.4.

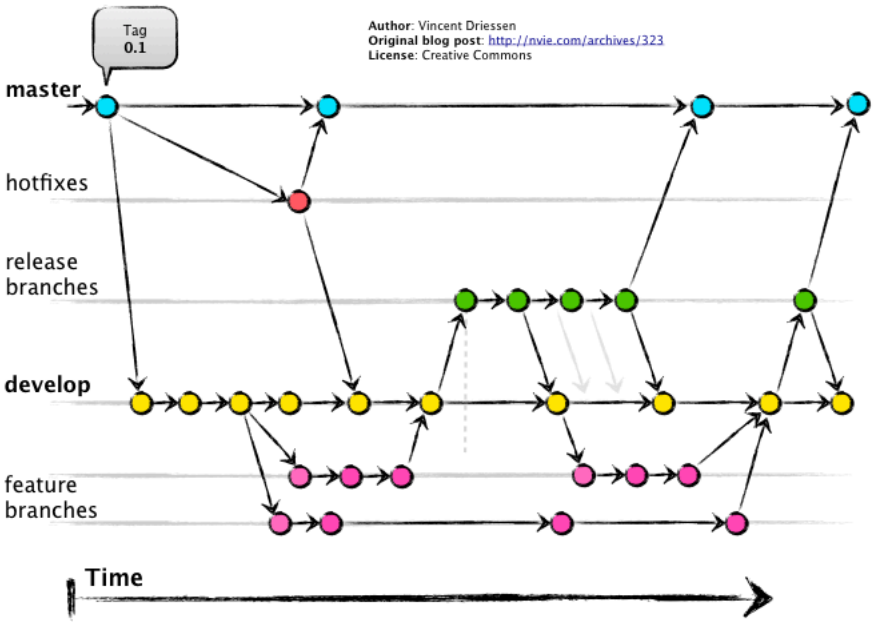


Figure 6.1: Gitflow branching example

Sprint 6

At this point, artists and event managers can configure artist or music event data, but having this structure implemented, adding functionalities to post custom notifications to show users what changed on a given artist or event they follow would increment the users interest in the application to keep updated about the events and artist they like.

To do this, the custom notification options where added to both artists and event administrators, which always include a custom description for them to post some text explaining what the notification is about (e.g. an artist confirming a tour date could say how glad they are to go there on their tour). Artists will be able to notify which tour dates they have confirmed or which they have cancelled. Event administrators will be able to notify artist attendance and artists cancelled who were previously confirmed as attendants, and ticket availability. Both artists and event administrators will be able to publish a generic custom notification, simply including some text, to provide general information (For example, an album release date).

Version control

This project followed **gitflow**, a branching model for Git which distinguishes types of branches, being the main ones, *develop* and *master*, and branches derived from them (*feature, bugfix, hotfix...*) to keep versions of the code organized. Gitflow provides support for emergency fixes and allows parallel development, although for a single person environment, this last feature was not taken advantage of. We can see a graphic example of gitflow in figure 6.1.

Chapter 7

Implementation and testing

This chapter will present a chronological view of the implementation and testing process following Scrum. As explained in the Methodology chapter, section 3.1, Scrum organizes the implementation in Sprints. Each Sprint will provide a deliverable product, adding an increment to the previous one. At the beginning of each Sprint, there is a planning of what will be implemented during it, and at the end, a review and retrospective to ensure the work planned for it is completed, which includes testing of the new features.

7.1 Sprints

For this project, the duration of every Sprint was 2 weeks, having a total of 6 Sprints.

Sprint 1: Application structure

This first sprint took place between 21th of October and 1st of November 2019.

The first week of this Sprint was entirely dedicated to create the basic structure for the front-end and back-end of the application, including some basic templates for the user interface, creating the database with the necessary entities and getting familiarized with Angular.

Once the skeleton of the application was done, the main structure for registering and logging in and out of the application was implemented for regular users (Artists and Event administrator users were not added until the artist administration and event administration began).

Front-end login method from *AuthenticationService.ts* is showed:

```
login(username: string, password: string): Observable<User> {
        const bodyRequest = {
            "Name": username,
            "Password": password
        };
        return this.http.post<User>(this.userUrl + "Authenticate",
    bodyRequest)
            .pipe(map(user => {
                // store user details and jwt token in local
    storage to keep user logged in between page refreshes
                localStorage.setItem('currentUser',
    JSON.stringify(user));
                this.currentUserId = user.id;
                this.currentUserType = user.role;
                return user;
            }));
    }
```

At the top right corner of the screen a button will display, as pop-up, a small log-in form with a link to a register page, which includes options for future registration as Artist or Event Manager.

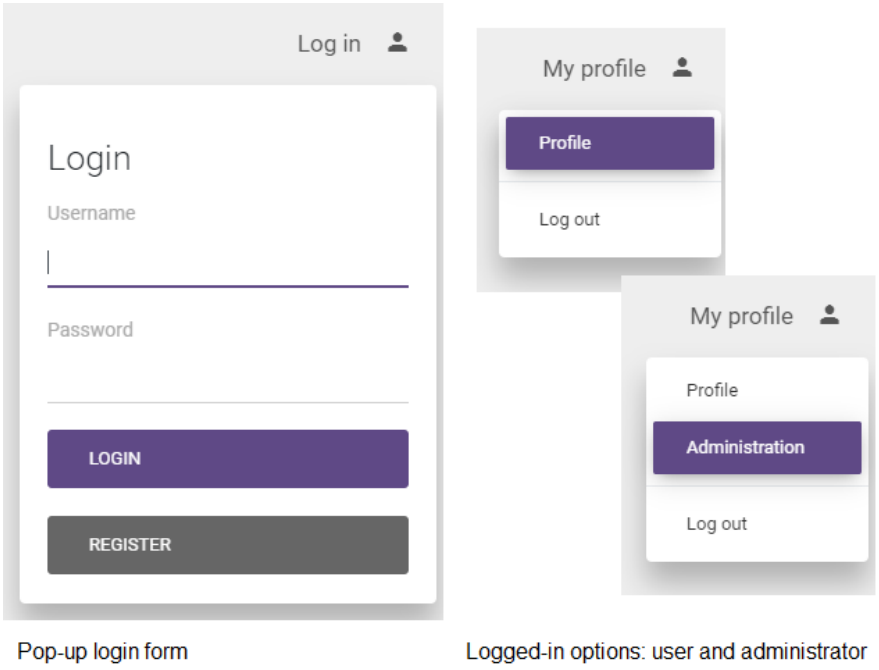
Once a user is registered and logged in the application, they can update their profile data through a profile page in order to change their login name, email, password and all information related to them. The final result of these login and logout options is showed in figure 7.1.

Sprint 2: Searching and filtering

This first sprint took place between 4th and 17th of November 2019.

To implement the Search User Stories 4.4.1, the basic layout for events and artists list and their filters to find certain criteria where added to the application, as it's shown in figure 7.2. These filters work as back-end filters, as loading all events or artists when loading the application would provoke a big delay and it's not worth it for filtering them at front-end, while searching only in a portion of the events would not extract desirable results for the user. The front-end initially loads the most popular music events and artists when showing the search pages.

The interfaces provided to search events and artists show a list of results which display only the most relevant information, to allow users to easily compare them: Music event list displays the music event's poster (which allows a fast lookup at artist attending the event), name, start and end date, type of event (festival, concert...) and the minimum price of available



Pop-up login form

Logged-in options: user and administrator

Figure 7.1: Login and logout pop-ups

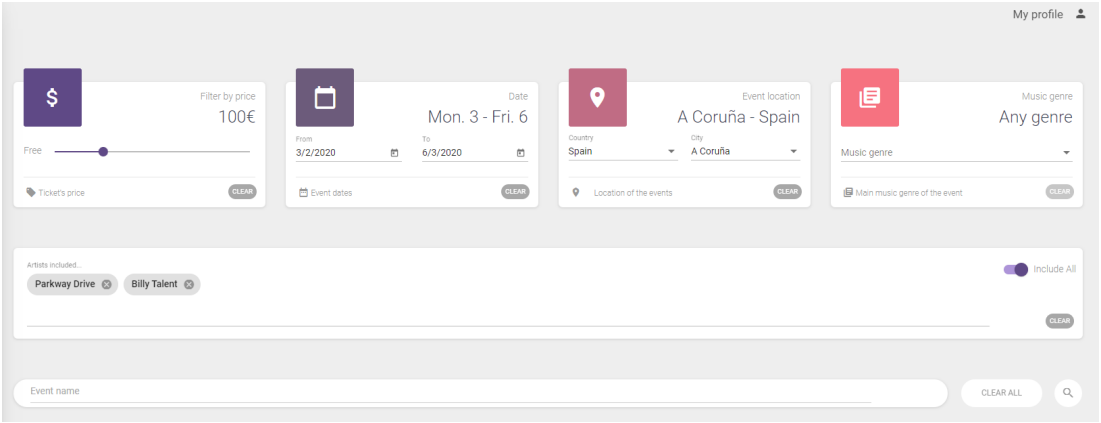


Figure 7.2: Filters for event search

tickets (if any). Artist list displays avatar, name, genres and number of tour dates (if any). The Back-end sends DTOs of this entities to the Front-end of the application, in order to avoid sending unnecessary attributes which are not displayed at these lists, such as, description or social media. This also allows adding fictional attributes which are not stored in the Database, for the details view of events and artists; Both of them include their number of followers. Artists include the number of tour dates (if any) and Music Events, the lowest ticket price available (if any).

Artist and Music Event final DTOs, including all attributes for details page and list pages (Although at this point in the development, social media and other details where not included, the final DTOs for these entities are showed, to avoid later repetition):

```
public class ArtistDto
{
    public int id { get; set; }
    public string name { get; set; }
    public string genre { get; set; }
    public string description { get; set; }
    public string city { get; set; }
    public string country { get; set; }
    public string logo { get; set; }
    public int countEvents { get; set; }
    public int followers { get; set; }
    public string type { get; set; }
    //Social media:
    public string twitter { get; set; }
    public string facebook { get; set; }
    public string instagram { get; set; }
    public string webpage { get; set; }
    public string spotify { get; set; }
}

public class MusicEventDto
{
    public int id { get; set; }
    public string name { get; set; }
    public DateTime startDate { get; set; }
    public DateTime endDate { get; set; }
    public Nullable<int> price { get; set; }
    public string typeOfEvent { get; set; }
    public string logo { get; set; }
    public string description { get; set; }
    public string location { get; set; }
    public string city { get; set; }
```

```
33      public string country { get; set; }
34      public int followers { get; set; }
35      //Social media:
36      public string twitter { get; set; }
37      public string facebook { get; set; }
38      public string instagram { get; set; }
39      public string webpage { get; set; }
40      public string sellpoint { get; set; }
41  }
```

To allow filtering events by artist attending them (all artist attending the event or any one of them, most as possible) the front-end will provide a chip-list selector of artists, which auto-completes text for artists names and displays a list of most popular artists.

Part of the *EventComponent.ts* corresponding to this artist chip-list's controller is showed:

```
1  artists: Artist[] = [];
2  allArtists: Artist[] = [];
3  artistCtrl: FormControl;
4  filteredArtists: Observable<any[]>;
5  readonly separatorKeysCodes: number[] = [ENTER, COMMA];
6  @ViewChild('artistInput', {static: false}) artistInput:
       ElementRef<HTMLInputElement>;
7  @ViewChild('auto', {static: false}) matAutocomplete:
       MatAutocomplete;
8
9  constructor(private eventService: MusicEventService,
10     private artistService: ArtistService,
11     private formBuilder: FormBuilder,
12     private masterService: CoreService) {
13
14     this.artistCtrl = new FormControl();
15     this.artistService.getAllArtists().subscribe(
16       artists => {
17         this.allArtists = artists;
18         this.filteredArtists = this.artistCtrl.valueChanges.pipe(
19           map((a) => a ? this._filter(a) :
       this.allArtists.slice()));
20       },
21       error => {
22         console.log(error);
23       }
24     );
25 }
26
27 private _filter(value): Artist[] {
28     //Value is the string the user writes until an artist is
```

```
    selected!
29      const filterValue = value.name? value.name.toLowerCase :
        value.toLowerCase();
30      return this.allArtists.filter(a =>
        a.name.toLowerCase().indexOf(filterValue) === 0);
31  }
```

To avoid a big amount of information sent to the client when setting the filtering criteria and searching, in case many results are found, the application provides paging and shows only the first 20 events following the users criteria. This way, the server's response is fast and also allows back-end filtering avoiding a large delay.

The SQL queries used by DAOs to obtain the filtered events add filtering criteria depending on which filters the user selects (all are optional and therefore are nullable at the server-side). DAOs create a SQL query depending on these filtering criteria and obtain the result to return to the client-side. These queries only make the necessary joins, to ensure they are not slowed down unnecessary. For example, a query which does not require any filters for ticket prices, will not join table Tickets.

Here we can see an example of an SQL query having the following filtering criteria: Events which include artists *Parkway Drive* and *Architects* (artists with Ids 1 and 2 respectively), whose name includes the "fest" word, where the main genre of the event is *metal* (genre with Id 3), which take place between 01/06/2020 and 30/06/2020 at Spain (country with Id 1) and which contain available tickets for a price lower than 125€. For this example, the page selected is the first one, with an element count of 20 music events per page, so the start index is not necessary while the count is set as 20 music events. This prevents the execution of the query to take too int if the Database contains lots of music events.

```
1  SELECT DISTINCT TOP 20 m.id, m.name, m.startDate, m.endDate,
       m.logo, typeOfEvent, MIN(t.price) AS price
2      FROM MusicEvent m
3      LEFT JOIN Ticket t ON m.id = t.eventId
4      LEFT JOIN ArtistEvent ae ON ae.eventId = m.id
5      LEFT JOIN ArtistGenre ag ON ae.artistId = ag.artistId
6      WHERE UPPER(m.name) LIKE '%FEST%'
7      AND t.price <= 125
8      AND m.startDate >= '2020-6-1'
9      AND m.endDate <=  '2020-6-30'
10     AND m.country = 1
11     AND ae.artistId IN (1,2)
12     AND ag.genreId = 3
13     GROUP BY m.id, m.name, m.startDate, m.endDate, m.logo,
       m.description, m.location, m.city, m.country, typeOfEvent,
       ag.genreId
```

```
    ORDER BY m.name
    HAVING COUNT(DISTINCT ag.artistId) = 2;
```

For a better database performance, nonclustered indexes are created on attributes which will be frequently used to search events or artists. For example, as events and artists will both be usually searched by name, the following indexes ease this search on the database (other indexes exist for ticket price, user login name, artist attendance to music events...):

```
CREATE NONCLUSTERED INDEX [IX_EventIndexByName]
ON [MusicEvent] ([name] ASC)

CREATE NONCLUSTERED INDEX [IX_ArtistIndexByName]
ON [Artist] ([name] ASC)
```

Sprint 3: Event and Artist followup

This first sprint took place between 18th of November and 1st of December 2019.

To implement the follow/unfollow artists and events User Stories, the N-M relationships between tables *User* and *Artist*, and *User* and *MusicEvent* were added as intermediate tables in the database *UserFollowsArtist* and *UserFollowsEvent*. The corresponding methods to follow/unfollow an artist or a music event were implemented.

Once a user has followed some artists or events, providing them a dashboard page with tabs for those artists and events they are interested into was a trivial functionality to add to the application. This new page was set as visible only to logged users, and was provided with two tabs: Artists and Music Events.

Detail pages for both artist and music events were implemented to show additional information to the basic one provided at the searching lists, such as, description, social media, list of tour dates for artists and list of attending artists and available tickets for music events. These detail pages were accessible through the search pages and the dashboard, adding in both of them, a link to each listed event and artist.

To implement the artist and event lists layout on the detail pages, previous components for listing artists and events were reused, receiving as input the artists attending an event (for music event details) or the tour dates beinting to an artist (for artist details). The available tickets list for a music event was implemented, as it was a new one, and was only going to be used at one place. These lists are accessible through tabs below the general information of a given artist or event.

HTML code corresponding to this new Ticket list, and the reused artist list:

```
 <!-- TICKETS -->
<div *ngIf="getSelectedTab('tickets') && hasTickets()">
```

```
  <div *ngFor="let ticket of tickets" class="card">
    <div class="card-header card-filter card-header-icon">
      <div class="card-icon">
        <h3>{{ticket.price€}}</h3>
      </div>
      <h3 class="card-category">{{ticket.name}}</h3>
      <p *ngIf="ticket.eventDate" class="card-description">
        {{ticket.eventDate | date}}
      </p>
      <p *ngIf="!ticket.eventDate" class="card-description">
        Full event
      </p>
      <p class="card-description">{{ticket.description}}</p>
    </div>
  </div>
</div>

<!-- ARTISTS -->
<div *ngIf="getSelectedTab('artists') && hasArtists()">
  <app-artists-list [artists]=artists></app-artists-list>
</div>
```

Sprint 4: Artist profile management

This sprint took place between 2nd and 15th of December 2019.

To implement the Artist Administration User Stories 4.4.3, the already existing Artist entity was extended in order to provide a reference to which user administrates it. This allows the use of a regular user entity, with it's login attributes, to also be used for an artist administrator, with the extra parameters the artist profile needs. For the Country and City relationship, which exists in both the User and the Artist entities, the Artist location is used, leaving the User's City and Country set to null, so that the location set by an Artist Administrator is available to locate the artist profile.

These additional functionalities for Artist administrators are provided at front-end as a new administration page to view all confirmed tour dates, with a cancellation button next to each one. To confirm new tour dates, the administrator will search music events by name and view them in a list with the additional confirm option. In case they search an already confirmed music event, it will appear as confirmed and this option will be disabled. In figure 8.11 of the Solution chapter, we can see an example of searching an event to confirm attendance to, after the name "Fest".

The already existing user profile page for editing user details will additionally include Artist information to edit their details, such as genres, description, avatar, etc.

The corresponding methods to confirm or cancel attendance to a music event were implemented. These methods receive the user Id from the User entity corresponding to the administrator logged in the application, and obtains the Artist entity they administrate through *Linq*.

Method from *ArtistService.cs* to confirm attendance to an event:

```
public void ConfirmTourDate(int eventId, int userId)
{
    Nullable<Artist> artist =
    _artistDao.FindArtistUserByUserId(userId);
    MusicEvent musicEvent = _musicEventDao.FindById(eventId);

    if (artist == null)
        throw new AdministrationPermissionsException();

    using (var context = new BDNetStageEntities())
    {
        //Found artist administrated by user with requested ID
        artist.AttendsEvent.Add(musicEvent);

        context.SaveChanges();
    }
}
```

Sprint 5: Event administrator profile management

This sprint took place between 16th and 29th of December 2019.

To implement the Event Administration User Stories 4.4.4, a new relationship between Users and Music events is required in order to allow one administrator to manage various events. Methods to create and update a Music event and its data were provided to those administrators through a new tab in the dashboard page of the application listing their administrated events, each of them, with a link to an administration page.

The administration page would provide a form to set all music event data (name, description, start date, end date, poster, location and social media).

To view attending artists and tickets, a list of confirmed artists and available tickets will be accessible through tabs. Both artist and ticket lists provide a remove button for each one. To confirm new attending artist, the administrator will search them by name and view them in a list with the additional confirm option. In case they search an already confirmed artist, it will appear as confirmed and this option will be disabled. To create a new available ticket, a ticket form will be filled with all ticket data (name, description, date and price), and it will be added to the list. Existing tickets can also be edited through a similar form.

The event administrator confirming method works similarly to the confirming tour date method for artist administrators, but as the user does not directly identify a single event (as event administrators can manage more than one event), this method needs the user, event and artist ids. To verify the user trying to perform this action has permission to do so, the music event relationship with administrator users is checked.

Method from *MusicEventService.cs* to confirm attendance to an event:

```
public void ConfirmArtistAttendance(int userId, int musicEventId,
    int artistId)
{
    Artist artist = _artistDao.FindById(artistId);
    MusicEvent musicEvent = _musicEventDao.FindById(musicEventId);

    if (musicEvent.AdministratedByUser.First().id == userId)
        using (var context = new BDNetStageEntities())
        {
            artist.AttendsEvent.Add(musicEvent);

            context.SaveChanges();
        }
    else
        throw new AdministratorPermissionsException();
}
```

Sprint 6: News and dashboard

This sprint took place between 30th of December 2019 and 12th of January 2020.

This sprint's selected User Stories were the ones regarding notifications for artist and music event updates, which were added to the Product Backlog during the development as new ideas to add value to the solution.

At this point, users may search, compare and follow events and artists, while administrators set their data, but if that data changes, users still have to check details of their followed artists and events in order to notice. To solve this, additional functionalities to provide notifications of this updates were added. A first thought was making automatic notifications whenever a tour date or artist attendance from an event was configured, but this was improved with customized notifications in which administrators can select what to notify, and set custom descriptions. These notifications would be available for artists and event managers in order to notify any changes they may confirm on their profiles, whether they are new tour dates in case of an artist profile, or new artists attending an event or ticket availability for an event manager. These notifications include a custom description to provide a closer relation between followers and artists or event managers, allowing explanations in case of cancella-

tions or general information about the confirmed tour dates, artists and tickets. In addition, a general notification is available for any general information administrators might want to publish.

From the user's perspective, notifications will display not only the descriptions and information published by the notification itself, but also some additional virtual information depending on their data. For example, for a user living in Spain, any notification from events or tour dates from artists he follows located on Spain will display an additional text saying those events are located in their country.

Notification options for **Artists** Tour date confirmations are provided at the tour dates list. For each tour date, a notify button will allow the artist to notify it, adding the custom description.

Method from *NewService.cs* to notify attendance to an event:

```
public void NotifyTourDate(int userId, int artistId, int eventId,
    string description)
    {
        Artist artist = _artistDao.FindById(artistId);

        if (artist.ArtistUser.First().id != userId)
            throw new AdministratorPermissionsException();

        using (var context = new BDNetStageEntities())
        {
            New confirmation = new New();
            confirmation.artistId = artistId;
            confirmation.eventId = eventId;
            confirmation.typeOfUpdate =
    NewTypes.ArtistConfirmedEvent;
            confirmation.description = description;
            confirmation.updateDate = DateTime.Now;
            context.New.Add(confirmation);

            context.SaveChanges();
        }
    }
```

Notification options for **Event managers** are provided at the confirmed artist and available ticket lists. Both lists will add checkbox selectors for each element (ticket or artist), allowing the administrator to make a selection of tickets or artists they want to notify. This allows showing only the artists or tickets they consider more interesting in the notification, and publish various notifications, as the information is usually published in small increments. Artists confirmed to attend an event are added among time, not all at once (first confirmation includes a couple headliners, later, a new confirmation adds some headliners and smaller

bands...). This notification options will allow publishing only the new confirmed artists selecting from them, the most relevant ones according to the event manager. Same happens with Tickets; usually, a full day pass is announced at a lower price, and when a given amount of tickets are sold, the next offer is announced. Later, day tickets are announced. Notifications for this ticket offers would allow the full day ticket offer to be updated with those price increments, adding the day offers later and removing any sold out offer.

Method from *EventService.cs* to notify artists attending the event:

```
public void NotifyArtistsAttendance(int userId, int musicEventId,
    list<int> artistIds, string description)
{

    LoggedUser user = _userDao.FindById(userId);
    MusicEvent musicEvent = _musicEventDao.FindById(musicEventId);

    if (!user.AdministratesEvent.Contains(musicEvent))
        throw new AdministratorPermissionsException();

    using (var context = new BDNetStageEntities())
    {
        New confirmation = new New();
        confirmation.eventId = musicEventId;
        confirmation.typeOfUpdate = NewTypes.EventConfirmedArtists;
        confirmation.description = description;
        confirmation.updateDate = DateTime.Now;

        context.New.Add(confirmation);

        foreach (int id in artistIds)
        {
            NewChanges confirmedArtist = new NewChanges();
            confirmedArtist.artistId = id;
            confirmedArtist.newId = confirmation.id;
            context.NewChanges.Add(confirmedArtist);
        }

        context.SaveChanges();
    }
}
```

News from Music events and Artists a logged user follows will be displayed as the default tab of the dashboard page. Artist and Music event details pages will also include a default tab with their News. These notification lists are formatted differently in a profile details page and the dashboard; at the dashboard they display the name of the Artist or Music event, and

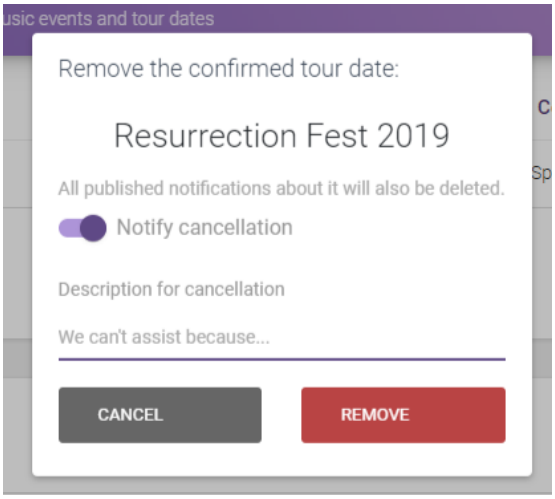


Figure 7.3: Confirmation when cancelling a tour date

the avatar or poster, respectively, while at the details page they don't, as the user is already viewing a specific Artist or Music event information, displayed in the details part above the News.

Every notification will have a remove option visible only for their administrators at their own profile (Artist profile or any Music event profile they manage, for an Event manager).

Cancellation notifications will also be available. When an administrator removes a tour date or confirmed artist, the pop-up confirmation for removing it will now add the option to publish a cancellation notification with its custom description. Cancelling a tour date will automatically remove any notifications about it, independently of notifying the cancellation or not. This can be seen in figure 7.3. Also, cancelling an artist from an event will remove them from notifications which included them (if they were the only artist displayed in the notification, it will be removed completely, but if not, the notification still remains with the other notified artists), independently of notifying the cancellation.

7.2 Testing

At the end of every sprint, the Sprint Backlog containing the User Stories selected to implement during it was reviewed to check the User Stories state and acceptance criteria. The User Stories are tested through black-box testing individually, and integration tests are performed on the complete solution. Black-box testing consisted on trying every feature and possible outcome of the application without specific knowledge of the application's code nor internal structure.

7.2.1 Unitary tests

Each User Story was tested individually, taking in consideration only aspects regarding it, through black-box tests. To ensure these tests are isolated, a testing database is used from scratch for each one of them. Testing each User Story and all possible outcomes required creating the necessary structure for the workflow, for example, to test deletion of a tour date successfully, creating it first is required. Following this approach, every action from regular users, artist users and event managers were tested at the end of each sprint.

7.2.2 Integration tests

Black-box tests were also used for integration testing, but this time, the whole workflow of the application was taken into account. These tests used a common database sharing data among them, verifying an action from one User Story would not break something in other one or create inconsistent data in the database. Each functionality and possible outcome from Users, Artists and Event managers actions were tested at the end of each sprint.

To simulate high demand situations, a database with over 1000 events and 1000 artists was created in order to test most SQL queries for searching and regular use of the application, by many users at the same time. As the amount of events or artists listed doesn't vary, the size of the database affects only the searching criteria, which proved to be fast enough even with complex SQL queries, with responses under 1s for very advanced searching criteria for concurrent requests over 100 users.

Chapter 8
Final solution

This chapter will show the final look of this application, and how it would be used by every type of user. Some screenshots have been cut to display only a particular table or element in order to display and fit them properly. As mentioned in section 5.4, the application provides different themes which change the color palettes of the interface. For simplicity, these interfaces will be showed with the main theme of the application, and the different themes will be showed only in some example screenshots in figure 8.1.

Searching and following solution

Search interfaces are shown in figures 8.2, 8.3 and 8.4. Event details page is shown in figure 8.5 and artist details page is shown in figure 8.6.

User functionalities

The user's profile configuration page is shown in figure 8.7 and the register or log-in page (as additional option besides the pop-up previously showed in figure 7.1), in figure 8.8.

Management functionalities

Artist management interfaces are shown in figures 8.9, 8.10 and 8.11. Event management interfaces are shown in figures 8.12, 8.13, 8.14, 8.15 and 8.16.

News and dashboard solution

Figure 8.17 show an example of notifications for registered users provided at the Dashboard page. Notifications displayed for a particular artist and event have already been shown in figures 8.5 and 8.6.

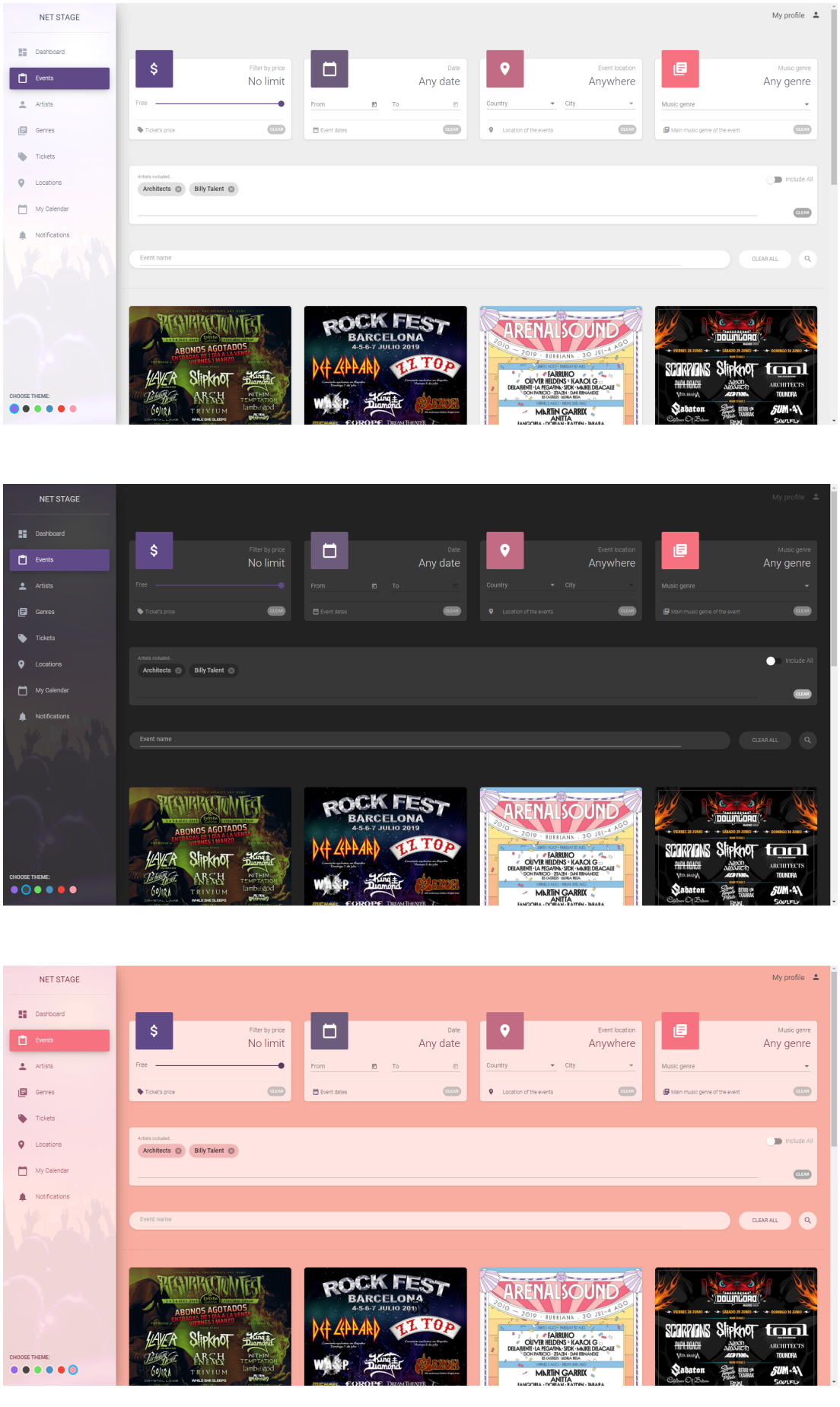


Figure 8.1: Default, dark and pink themes example pages

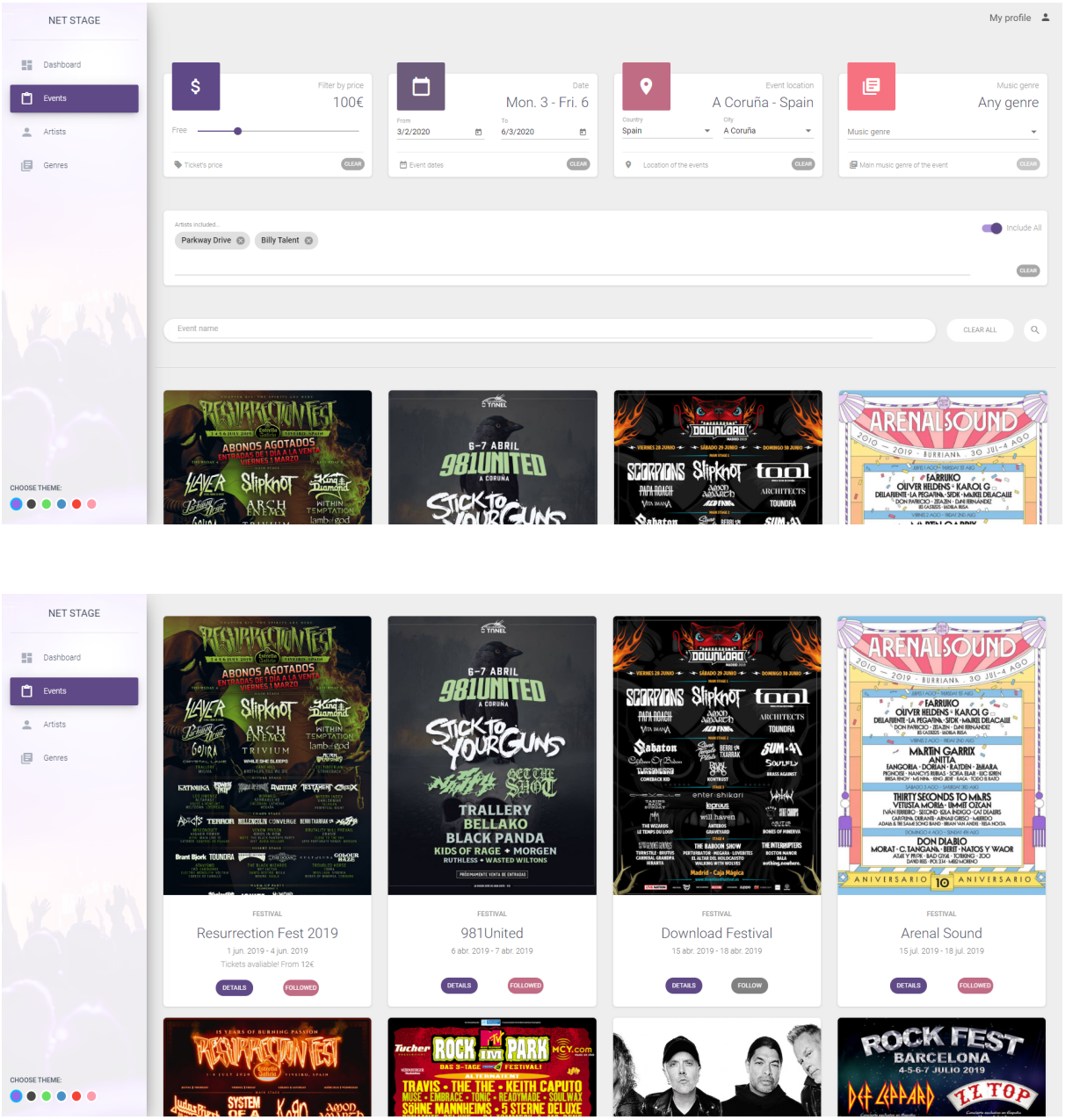


Figure 8.2: Search and follow music events page

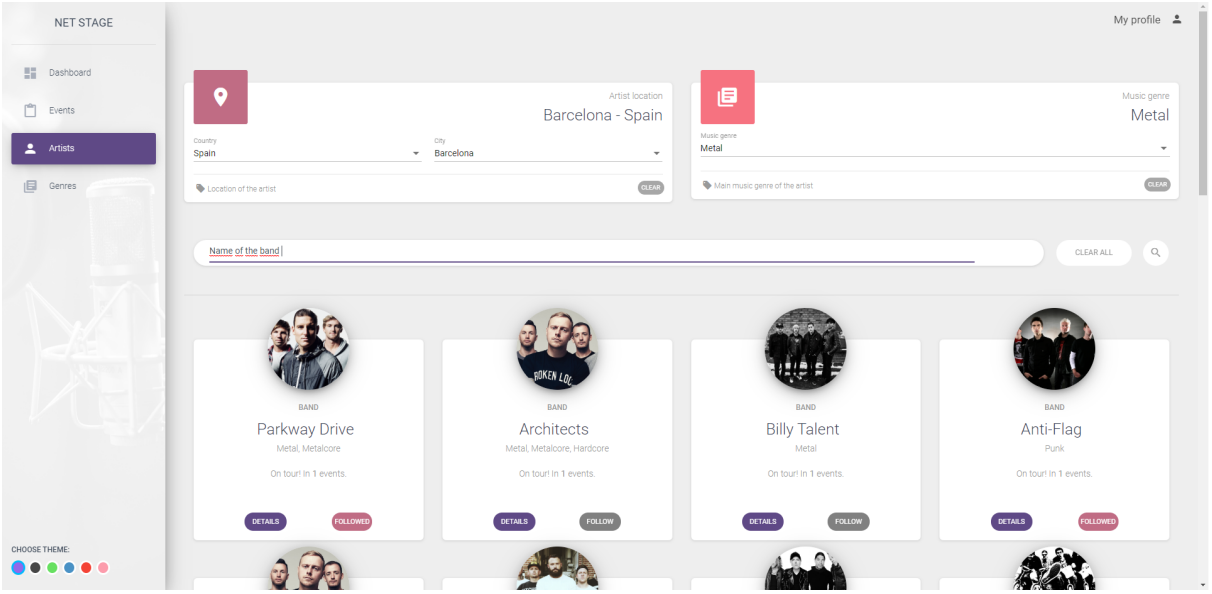


Figure 8.3: Search and follow artists page

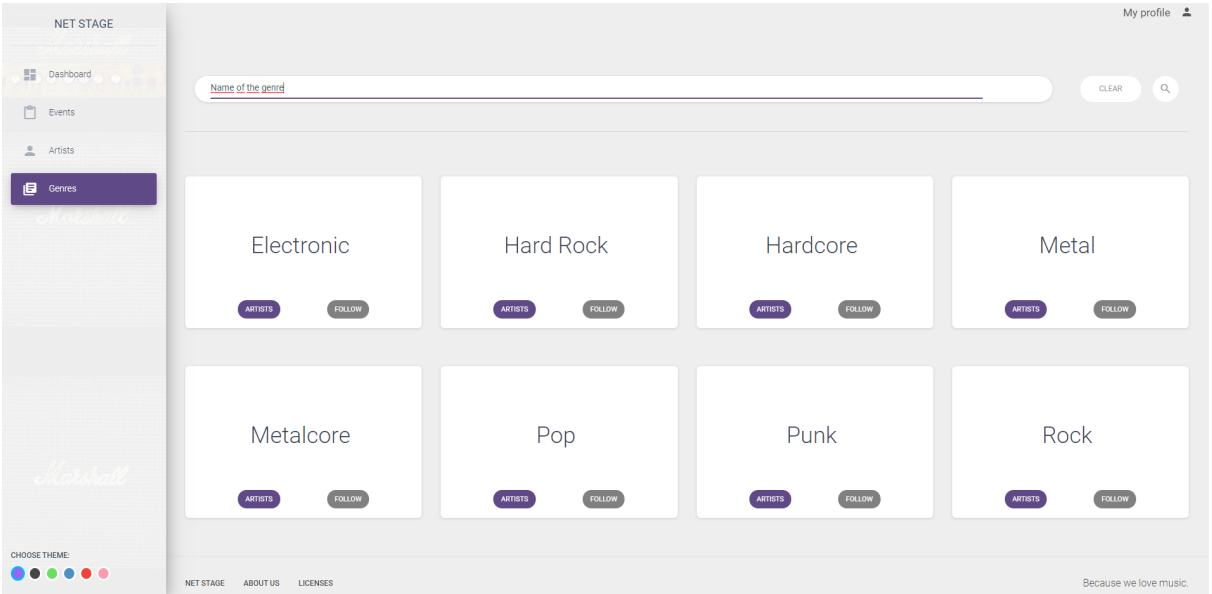
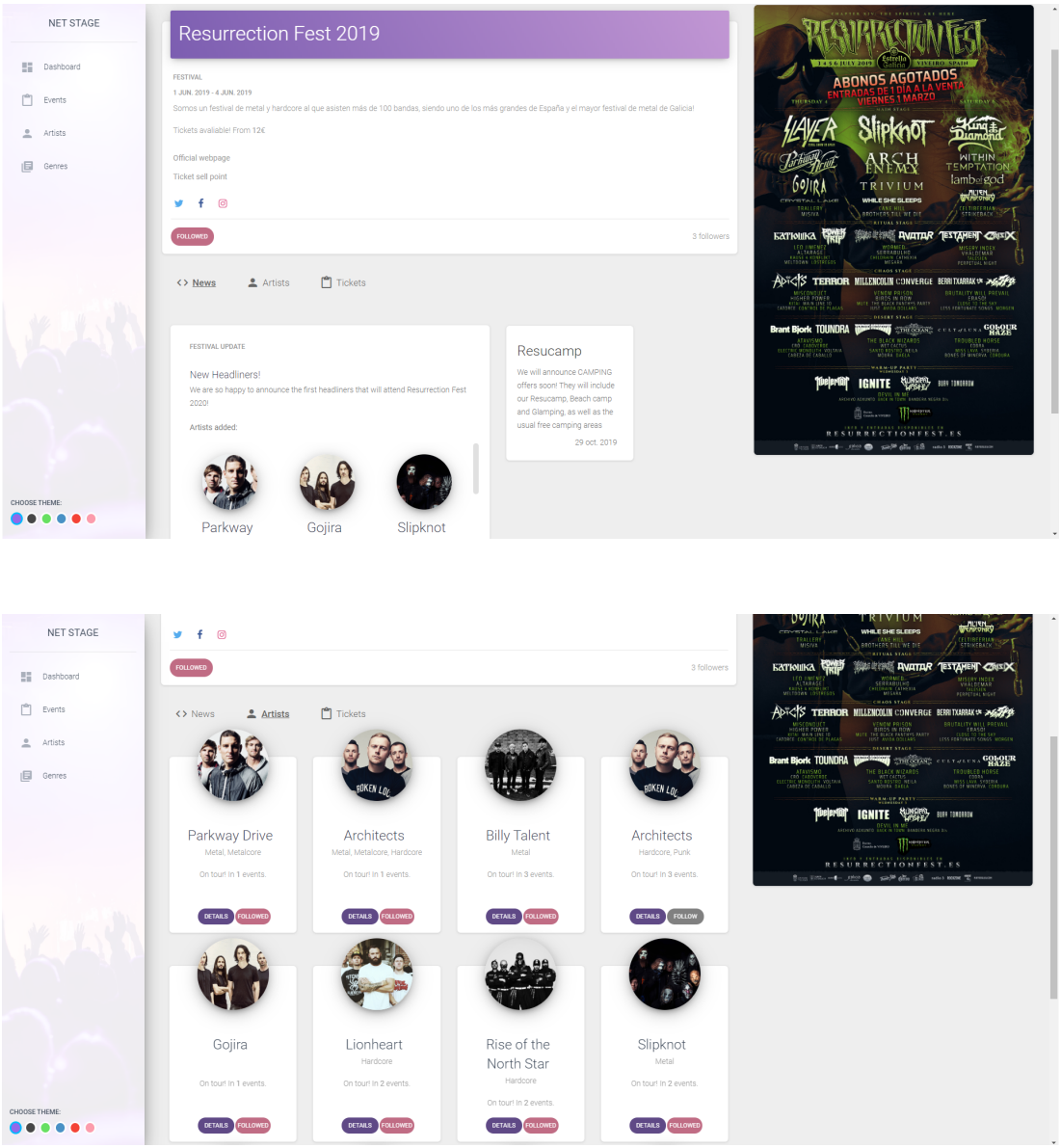


Figure 8.4: Search and follow genres page



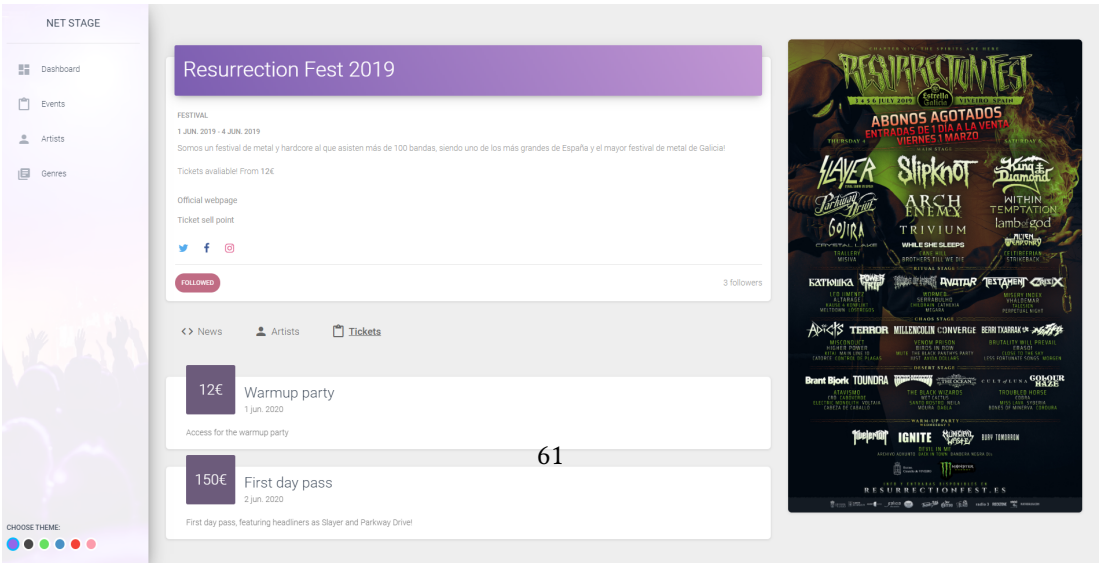


Figure 8.5: Music event details page example (News, artists and available tickets)

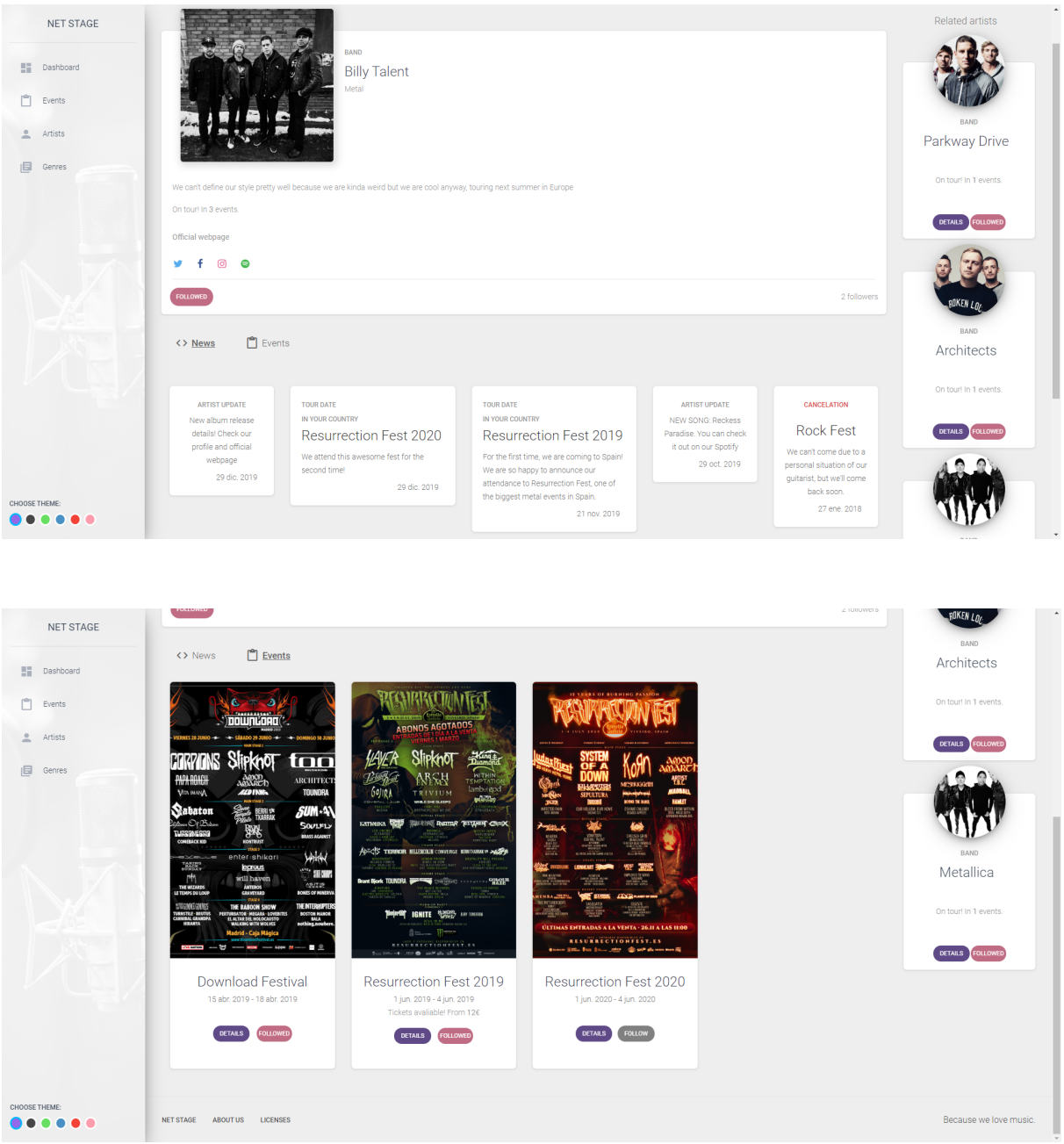


Figure 8.6: Artist details page example (News and events)

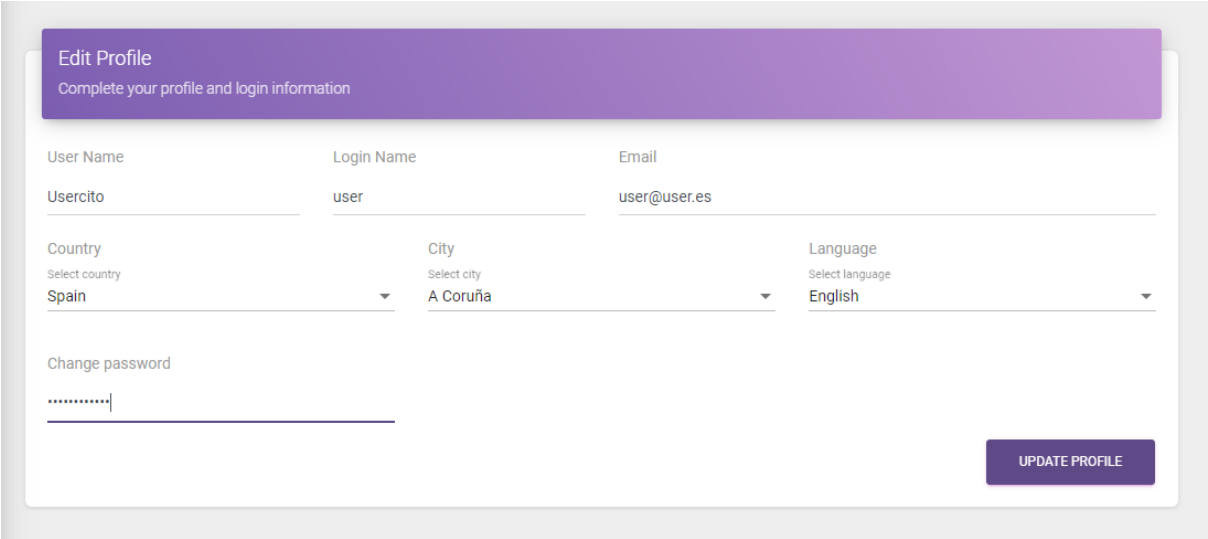


Figure 8.7: User profile configuration page

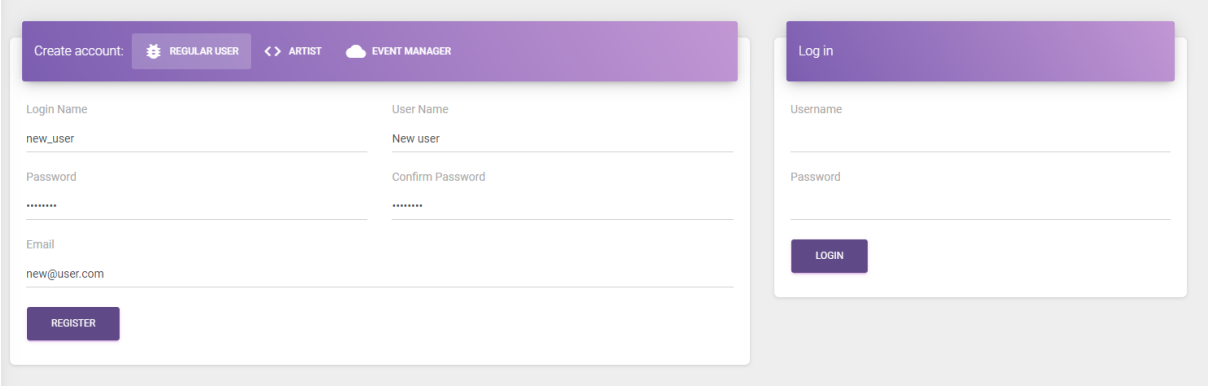


Figure 8.8: Register or log in page

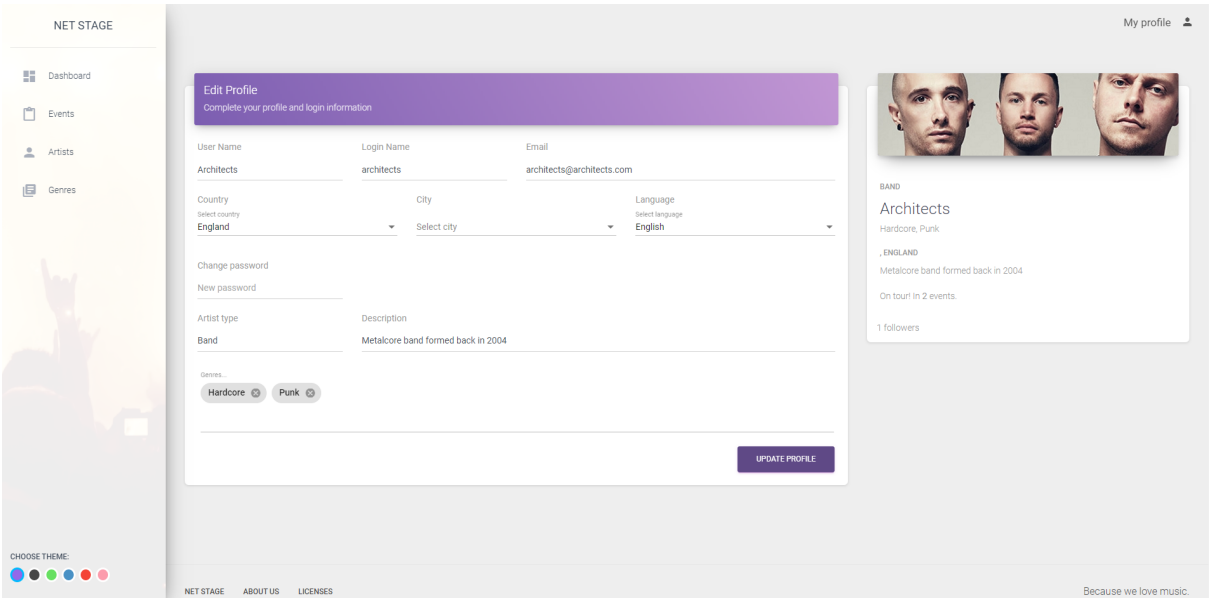


Figure 8.9: Artist profile configuration page

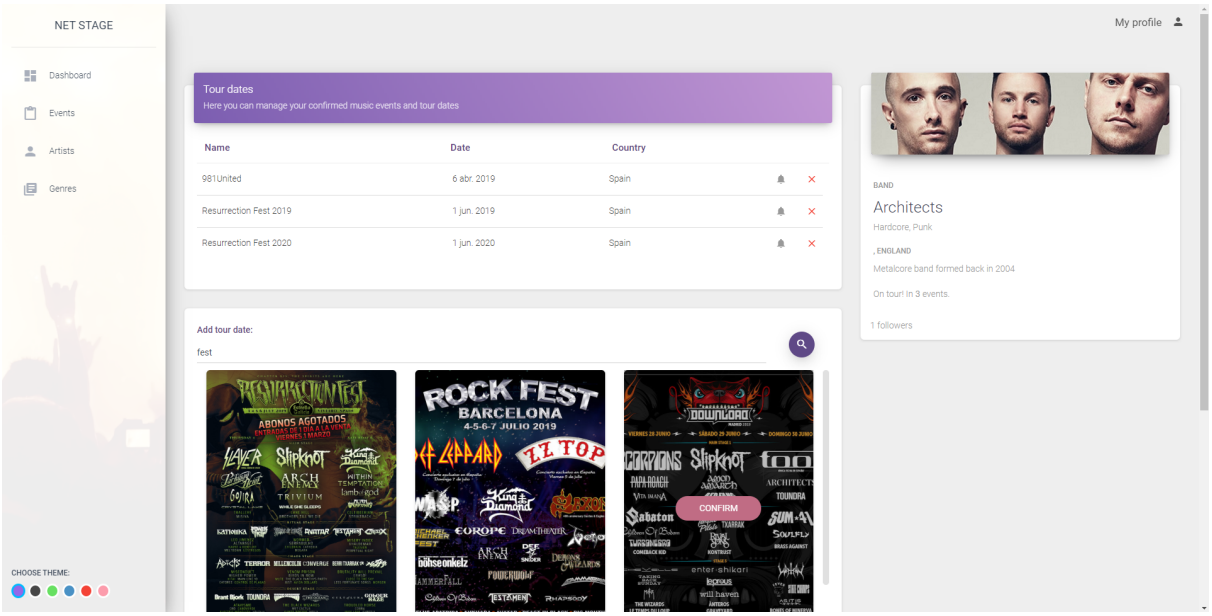


Figure 8.10: Artist administration page

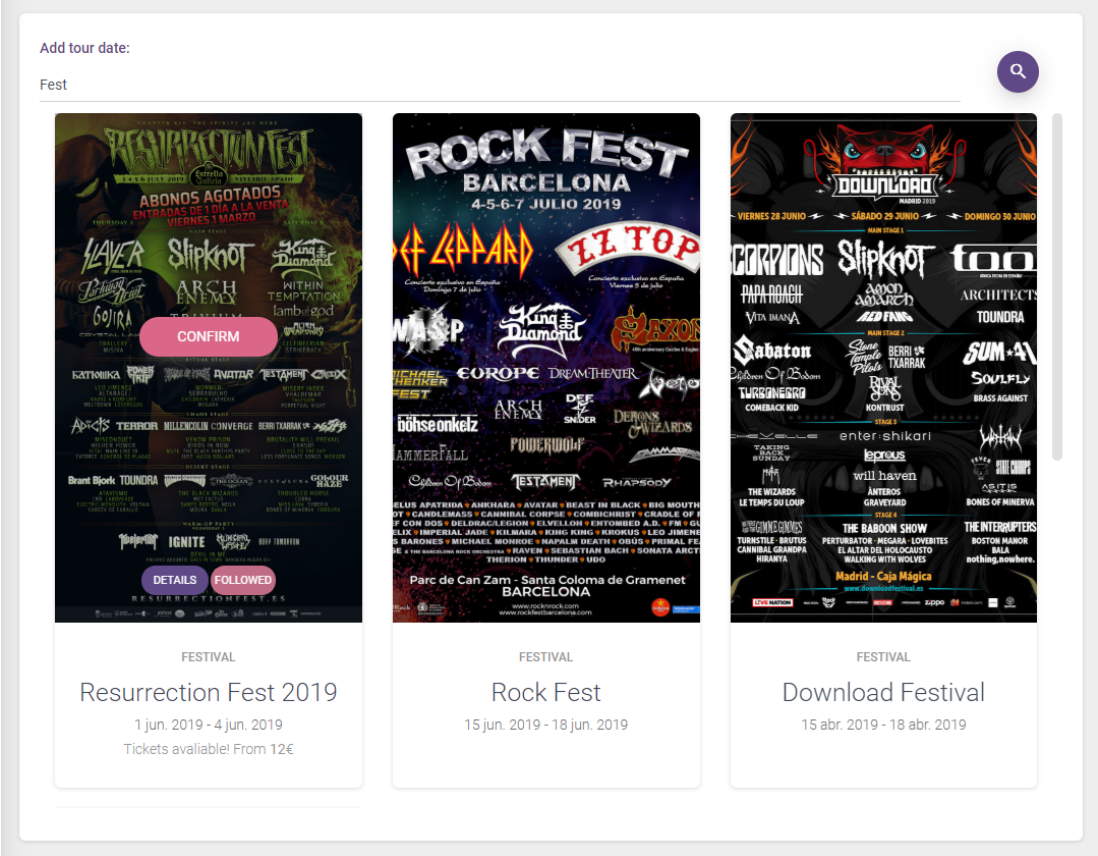


Figure 8.11: Searching a music event to confirm a tour date

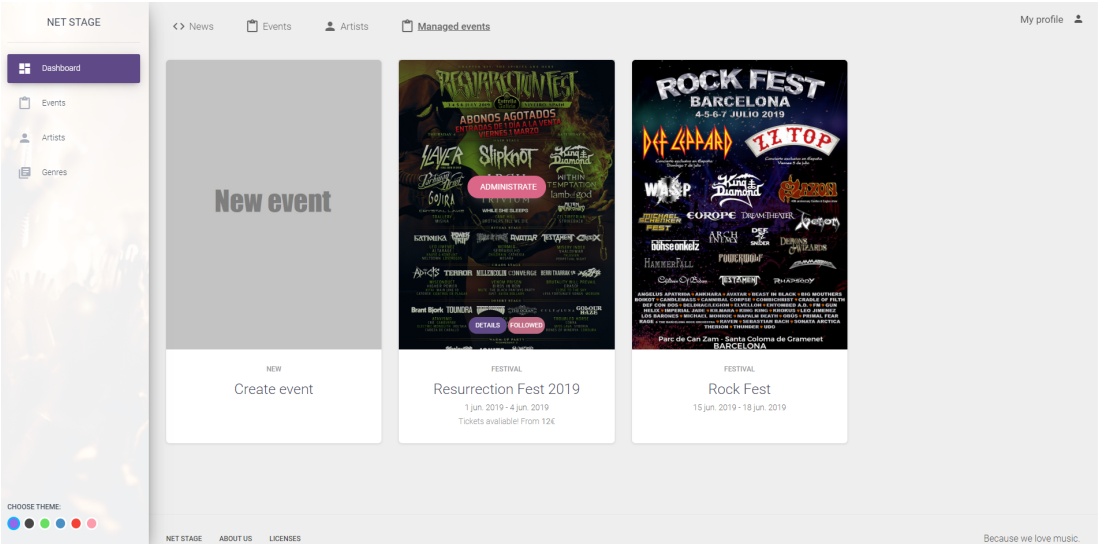


Figure 8.12: Music events an Event manager can configure

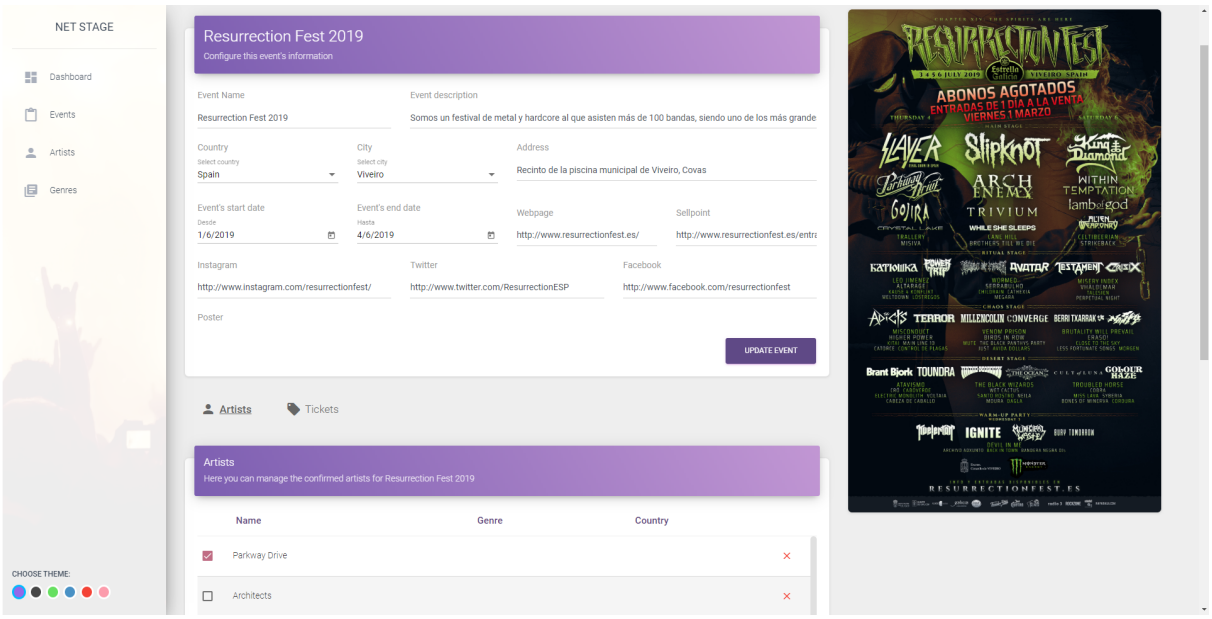


Figure 8.13: Music event administration page

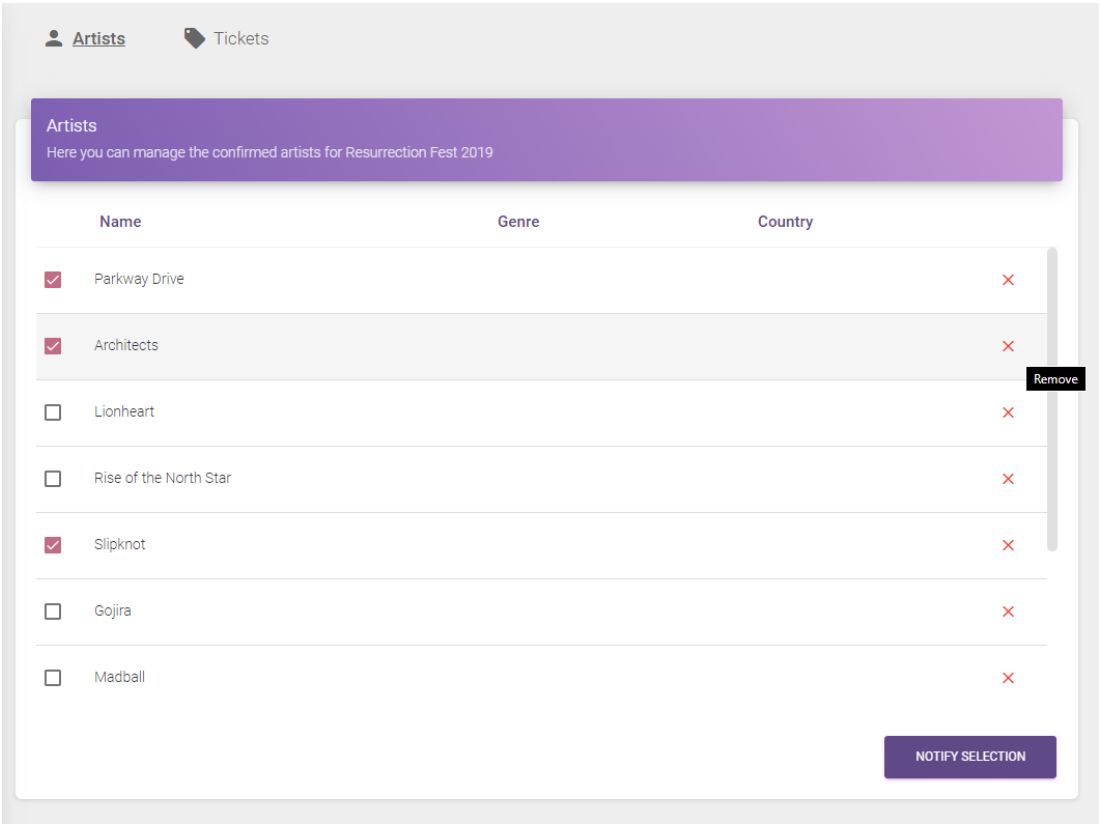


Figure 8.14: Artists attending a music event.

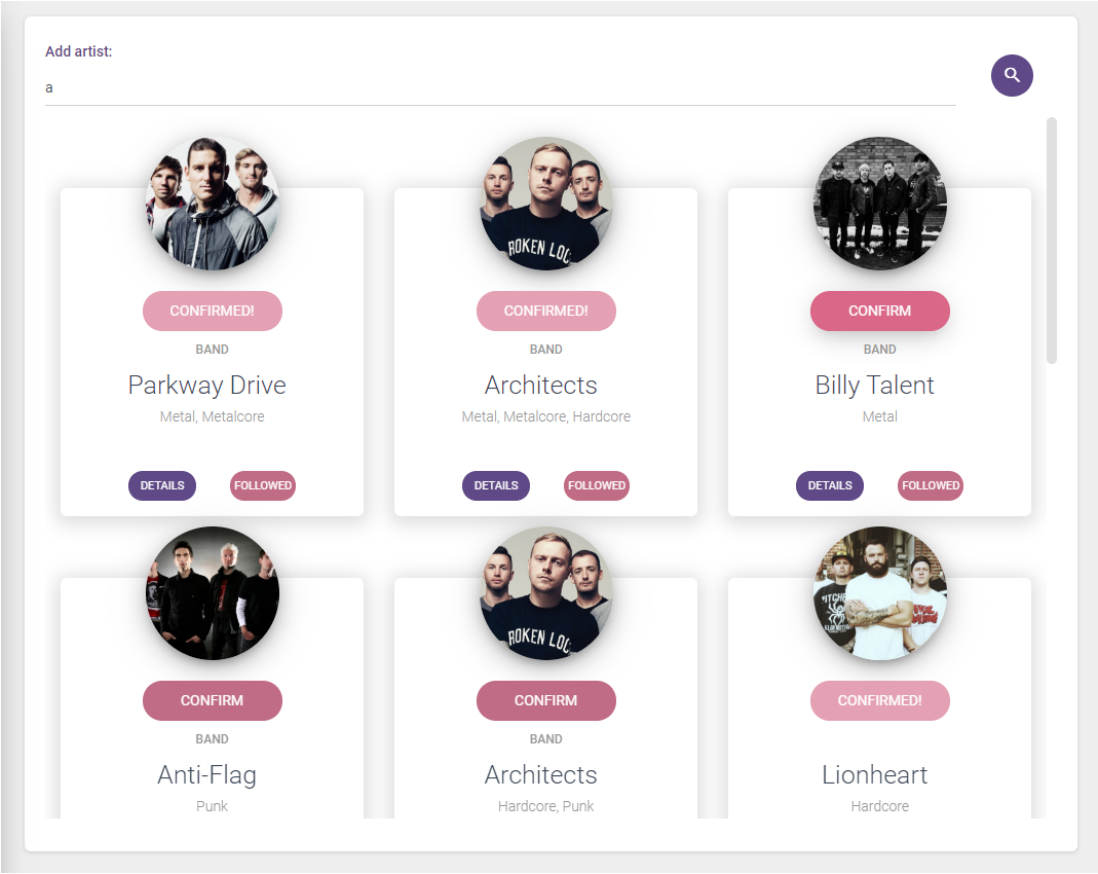


Figure 8.15: Searching an artist to confirm their attendance to an event

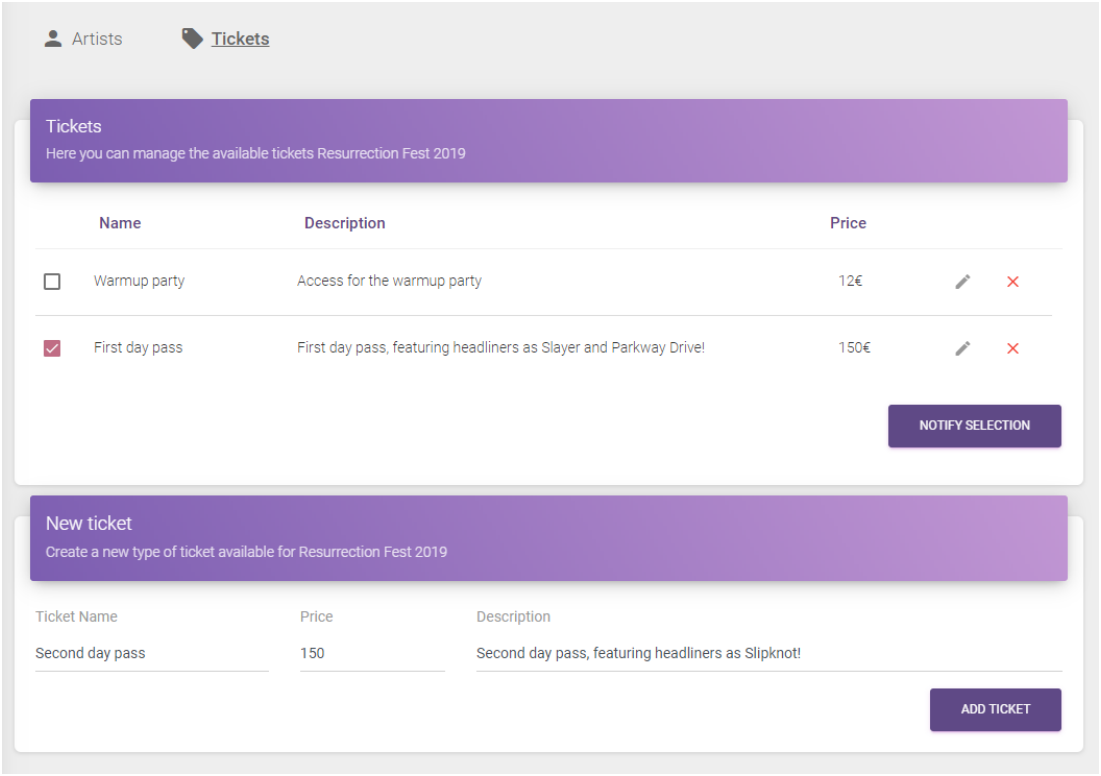


Figure 8.16: Tickets of a managed event, and the create ticket form

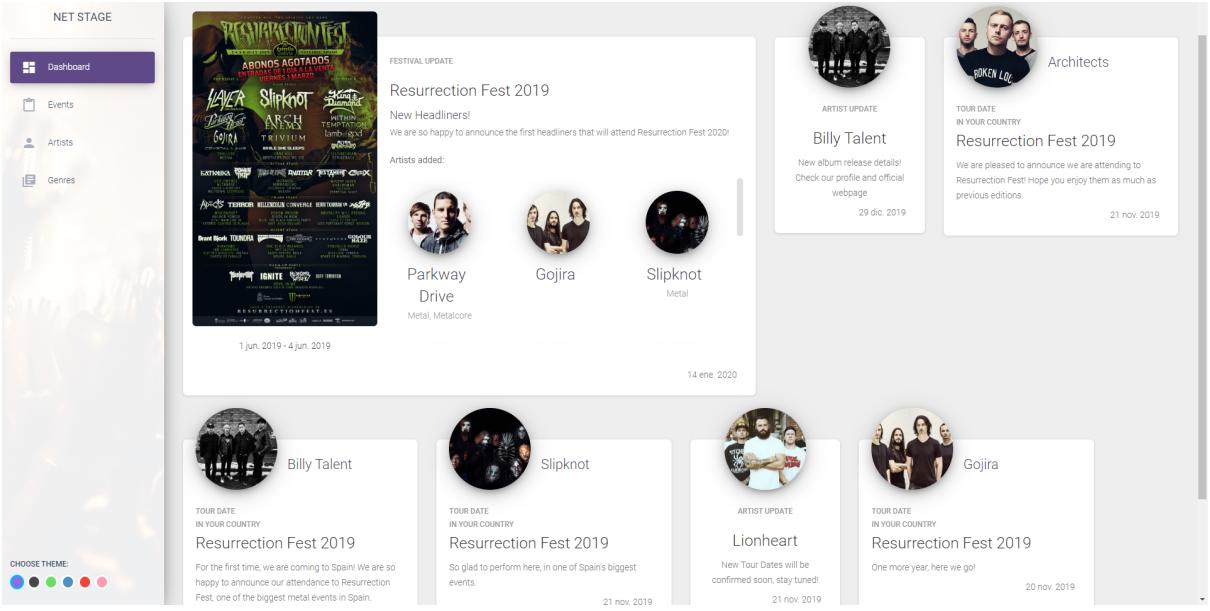


Figure 8.17: Dashboard notifications from events and artists

Chapter 9

Conclusions and future work

This chapter will cover the state of the project at the time of writing this report, lessons learned during this development, future improvements for the application and functionalities that could be added.

9.1 Conclusions

The result at the moment of writing this report was a consistent and robust application for music event and artist search and comparison, covering administration of all searching criteria and further details of artists and music events. Users are provided with relevant information and searching criteria while administrators can update their information and notify it keeping users updated. Time responses of the application proved to be fast even at high demand situations.

The user interface implemented ensures scalability, as we have seen when adding custom components such as artist or event lists on different parts of the application. Angular also allowed in depth customization of components, creating and lay-outing every page of the application in a fully customized way.

Regarding the methodology and the implementation phase, new ideas and features were added enlarging and adding more value to the final product. Custom notifications were a big feature added during the development, which provides cohesion between users and administrators. Other small features were also polished along time.

9.1.1 Lessons learned

The methodology for this development was familiar to the student from the subject *Metodoloxías de Desenvolvemento*, although it had to be adapted to a single person environment. .NET framework and Linq were also familiar as they were studied in *Marcos de Desenvolvemento*, but an in-depth learning was required in order to develop the application.

Previous knowledge in REST applications, design patterns and .NET framework obtained from subjects as *Ferramentas de Desenvolvemento*, *Marcos de Desenvolvemento* and *Internet e Sistemas Distribuídos* set the basis for the back-end development. The server-side development consolidated that knowledge and also refreshed the importance of good practices. The data design, access and optimization for the application through SQL was entirely based on SQL knowledge obtained on *Bases de Datos* and *Bases de Datos Avanzadas*.

The client-side development allowed learning and obtaining a good basis for front-end development. Angular framework was almost completely new for the student, and proved to have a comfortable and friendly learning curve, allowing implementation of an excellent user interface. Subjects as *Interfaces Home Máquina* provided a base for lay-outing the interfaces, which was expanded through the development.

9.2 Future work

Additional functionalities, ideas and extensions of the application raised during the development but weren't able to be implemented for this project. They will be presented here, along with reasons they could not be added to the current implementation.

Calendar view

It would be interesting to have an option of viewing events of the users interest, in a calendar format. This would be an additional option besides following, named "add to calendar", which would allow the user to select which music events they want to see in a personal calendar with dates of those events.

Locations

Other interesting way of looking for events or artists would be by their location. A map with events or artists pinned at their location would provide a nice view for the user to find artists and music events near them. These pins could show a small pop-up with the simplified information and a link to their details page.

Tickets

Expanding the Tickets of the application to provide sell point and allow saving tickets on a digital format would enhance the application in this area, which is not as deeply covered for this project.

The selling tickets option was discarded during the development due to its complexity and because most events have few official sell points and it would be difficult to obtain permission

to become one of them.

The digital format of the users already bought tickets could be added to the application as a digital ticket wallet in the future.

User comments

Allowing users to post comments on artists and music events would provide extra information for other users interested on them. Comment responses could also be added, and administrator responses could resolve doubts on events or tour dates, making their relationship with the users much closer.

Web scrapping

Web scrapping, web harvesting, or web data extraction is data scraping used for extracting data from websites. This means obtaining data from external websites automatically, typically using a web crowler. This could be used to complete information about artists and events from other pages, to enlarge the applications database with other data besides the one administrated by Artists and Event managers.

This is considered as a future upgrade to the application, but was not covered in this development because the data required to collect for this application is not easily found at the same web page, and web scrapping requires more work if the data is retrieved from different web pages and in different formats.

List of Acronyms

IoC *Inversion of Control.*

REST *Representational State Transfer.*

CRUD *Create, Read, Update, Delete.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

HTTP *Hypertext Transfer Protocol.*

DAO *Data Access Object.*

DTO *Data Transfer Object.*

MVC *Model View Controller.*

DB *Database.*

Bibliography

[1] Beatmash Magazine, "Número de asistentes festivales de españa 2018," 2018. [Online]. Available at: https://www.beatmashmagazine.com/numero-asistentes-festivales-espana-2018/

[2] S. Millett, *Professional ASP.NET Design Patterns*, 1st ed. Sons Inc, 2010.

[3] W. Penberthy, *Exam Ref 70-486: Developing ASP.NET MVC 4 Web Applications*, 1st ed. Microsoft Press, 2013.

[4] B.-A. GUÉRIN, *ASP.NET con C# en Visual Studio 2017. Diseño y desarrollo de aplicaciones Web*, 1st ed. Prentice Hall, 2018.

[5] M. Gruber, *Mastering SQL*. SYBEX, 2000.

[6] D. Petkovic, *Microsoft SQL Server 2019: A Beginner's Guide*, 7th ed. McGraw-Hill Education, 2020.

[7] "Indexes." [Online]. Available at: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver15

[8] C. N. D. U. Chandermani Arora, Kevin Hennessy, *Building Large-Scale Web Applications with Angular: Your one-stop guide to building scalable and production-grade Angular web apps*, 1st ed. Packt Publishing, 2018.

[9] "Introduction to the angular docs." [Online]. Available at: https://angular.io/docs/

[10] M. R. Gómez, *Curso de Desarrollo Web: HTML, CSS y JavaScript*, 1st ed. rupo Anaya Publicaciones Generales, 2017.

[11] V. K. Kotaru, *Angular for Material Design: Leverage Angular Material and TypeScript to Build a Rich User Interface for Web Apps*, 1st ed. Apress, 2019.

[12] "Angular material." [Online]. Available at: https://material.angular.io/

[13] B. Cherny, *Programming TypeScript: Making Your JavaScript Applications Scale*, 1st ed. O'Reilly UK Ltd, 2019.

[14] G. Verheyen, *Scrum - A Pocket Guide (Best Practice (Van Haren Publishing))*, 1st ed. Van Haren Publishing, 2013.

[15] A. Stellman, *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*, 1st ed. O'Reilly Media, 2013.

[16] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, 1st ed. Prentice Hall, 2017.

[17] ——, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Prentice Hall, 2008.

[18] M. Nayrolles, *Angular Design Patterns: Implement the Gang of Four patterns in your apps with Angular*, 1st ed. Packt Publishing, 2018.